НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «*Комплекс задач з підтримки процесу управління командою виконавців*»

Виконала: студентка *IV* курсу, групи *ІС-63*

Авраменко Катерина Анатоліївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Жданова Олена Григорівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *ст.вик., Проскура Світлана Леонідівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц., к.т.н., доц. Репнікова Наталія Борисівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Авраменко Катерині Анатоліївні
(прізвище, ім’я, по батькові)

1. Тема проєкту «Комплекс задач з підтримки процесу управління командою виконавців»

керівник проєкту *Жданова Олена Григорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використання

3. Креслення вигляду екранних форм

4. Схема бази даних

5. Схема структурна частини АС

6. Схема структурна послідовності

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1. Загальні положення	Жданова Олена Григорівна, к.т.н., доцент		
2. Інформаційне забезпечення	Жданова Олена Григорівна, к.т.н., доцент		
3. Математичне забезпечення	Жданова Олена Григорівна, к.т.н., доцент		
4. Програмне та технічне забезпечення	Жданова Олена Григорівна, к.т.н., доцент		
5. Технологічний розділ	Жданова Олена Григорівна, к.т.н., доцент		

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*14.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*15.04.2020*	
3.	*Постановка та формалізація задачі*	*17.04.2020*	
4.	*Розробка інформаційного забезпечення*	*20.04.2020*	
5.	*Алгоритмізація задачі*	*22.04.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*22.04.2020*	
7.	*Розробка програмного забезпечення*	*30.04.2020*	
8.	*Налагодження програми*	*07.05.2020*	
9.	*Виконання графічних документів*	*10.05.2020*	
10.	*Оформлення пояснювальної записки*	*12.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Катерина АВРАМЕНКО

Керівник Олена ЖДАНОВА

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6101.00.000 ПЗ	Пояснювальна записка	76	
3	А4	ДП 6101.01.000 ТЗ	Технічне завдання	8	
4	А3	ДП 6101.02.000 ССД	Схема структурна діяльності	1	
5	А3	ДП 6101.03.000 ССВ	Схема структурна варіантів використання	1	
6	А3	ДП 6101.04.000 КЕ	Креслення вигляду екранних форм	1	
7	А3	ДП 6101.05.000 СБД	Схема бази даних	1	
8	А3	ДП 6101.06.000 САС	Схема структурна частини АС	1	
9	А3	ДП 6101.07.000 ССП	Схема структурна послідовності	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6101 00.000.00		
Розробн.		Авраменко К.А.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Жданова О.Г.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н/контр.		Проскура С.Л.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Комплекс задач з підтримки процесу управління командою виконавців

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з шести розділів, 16 рисунків, 23 таблиці, 2 додатки, 6 графічних матеріалів, 18 джерел.

Дипломний проєкт присвячений розробці комплексу задач з підтримки процесу управління командою виконавців. Метою даного комплексу задач є підвищення ефективності роботи команди виконавців, за рахунок максимального дотримання директивних строків та мінімізації середнього часу виконання робіт.

У розділі інформаційного забезпечення визначені вхідні та вихідні документи для комплексу задач. Виконано проєктування бази даних, з детальним описом таблиць, полів та зв'язків між таблицями.

Розділ математичного забезпечення присвячений наближеному алгоритму для складання розкладу виконання робіт з різними директивними строками ідентичними машинами за двома критеріями.

У розділі програмного забезпечення проведено огляд технологій, що використовуються для розробки Комплексу. Розглянута архітектура комплексу, основні сутності. Було наведено основні прикладні програмні інтерфейси.

У технологічному розділі надано керівництво користувача для взаємодії з комплексом та проведено тестування функцій.

ДИРЕКТИВНІ СТРОКИ, НУЛЬОВЕ ЗАПІЗНЕННЯ, СУМАРНА ТРИВАЛІСТЬ ВИКОНАННЯ, СКЛАДАННЯ РОЗКЛАДУ, SCRUM, КОМАНДА ВИКОНАВЦІВ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of six sections, contains 16 figures, 23 tables, 7 appendices, 18 sources.

The diploma project is devoted to the development of a set of tasks to support the management process of the team of performers. The purpose of this set of tasks is to increase the efficiency of the team of performers, due to maximum compliance with directive deadlines and minimization of the average time of work.

The section of information support defines input and output documents for a set of tasks. Database design, with detailed description of tables, fields and relationships between tables.

The section of mathematical support is devoted to the approximate algorithm for making the schedule of performance of works with various directive terms for identical cars on two criteria.

The software section provides an overview of the technologies used to develop the Complex. The architecture of the complex, the main essences are considered. The main application programming interfaces were given.

The technological section provides a user guide for interaction with the complex and tests of functions.

DUE DATES, ZERO DELAY, TOTAL DURATION, SCHEDULING, SCRUM, TEAM.

ЗМІСТ

ВСТУП

Час – найбільш цінний ресурс, який людина може мати. Ви ніколи не зможете отримати більше часу та коли час використаний, його вже не повернеш. В сучасному світі вкрай важливо ефективно використовувати цей ресурс, недарма кажуть «час – це гроші». Такої думки притримуються і власники підприємств та корпорацій, адже в бізнесі раціональне використання часу може допомогти збільшити конкурентоспроможність компанії на ринку. Цим пояснюється популярність застосування задач складання розкладів та впорядкуванням робіт з метою покращення загального часу виконання на підприємствах.

Зосередимось на сфері інформаційних технологій. Для управління часом та процесом розробки існує багато підходів, серед них неітераційний, ітераційний тощо. Розробка програмного забезпечення неітераційним способом в більшості випадків є причиною того, що багато проєктів з висококваліфікованою командою є неефективними, або провальними. Проблема таких підходів полягає в тому, що коли встановлюється довгостроковий план, команда йому слідує і певним чином втрачається можливість гнучких змін, адаптації проєкту під початкові вимоги клієнта. Щоб уникнути цієї проблеми було запропоновано ітераційний підхід, за якого процеси планування, розробки, тестів, випуску розбиваються на короткі проміжки часу (як правило, тижні). Це не тільки дозволяє клієнту почати використовувати програмне забезпечення раніше (збільшуючи цінність для свого бізнесу), але й надає йому можливість швидшого зворотного зв'язку.

Для розробки програмного забезпечення команди все частіше застосовують гнучкі методології, які показали себе як високоефективні. Scrum – один з найбільш відомих та широко використовуваних ітераційних підходів управління проєктами. Зрозуміло, що є багато інших факторів, які впливають на успіх програмного проєкту, такі як якість коду, процес

випуску, архітектура тощо. Scrum не визначає найкращих практик програмної інженерії, він стосується управління проєктом програмного забезпечення.

Scrum простий: принципи, події, ролі інтуїтивно зрозумілі. Організація складних завдань у історії користувачів робить його ідеальним для складних проектів. Також чітке розмежування ролей та запланованих заходів забезпечує прозорість та колективну участь протягом усього циклу розробки. Швидкі релізи забезпечують мотивацію команди та задоволення користувачів, оскільки вони можуть побачити прогрес за короткий проміжок часу.

У Scrum визначені 3 основні ролі:

- власник проєкту: визначає пріоритетність задач, що складають резерв проєкту, відповідає на бізнес-питання, які може мати команда виконавців тощо;
- команда виконавців: складається з людей, які мають технічні знання та навички – це розробники, системні адміністратори, адміністратори баз даних тощо;
- скрам майстер (англ. Scrum Master): усуває можливі перешкоди, які можуть з’явитися для команди розробників на етапі розробки, допомагає проводити зустрічі Scrum тощо.

Одна з основних особливостей Scrum – самоорганізація команд. Вони не мають керівника. Саме тому, більшість процесів управління лягає на команду, а саме:

- розподіл та планування задач;
- відстеження стану задач;
- слідкування за своєчасністю виконання задач.

Дипломний проєкт присвячений розробці комплексу задач з підтримки процесу управління командою виконавців, який допоможе підвищити ефективність роботи команди виконавців, за рахунок максимального

дотримання директивних строків та мінімізації середнього часу виконання робіт.

Практичне значення одержаних результатів. Розроблено інструмент, який допоможе Scrum-командам здійснювати управління процесами та розробкою, запропоновано алгоритм розв’язання двокритеріальної задачі складання розкладу виконання робіт з різними директивними строками ідентичними машинами.

Публікації. Результати роботи були опубліковані у тезах доповідей на науково-технічній конференції [1].

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Є деяка обслуговуюча система (компанія), яка отримує замовлення на розробку програмних проєктів. Для управління процесом виготовлення проєкту використовується гнучка методологія розробки програмного забезпечення – Scrum. Scrum забезпечує чітку організаційну структуру, а також методи планування процесів.

На кожен проєкт компанія призначає команду виконавців (Scrum Team). Команди складаються з множини учасників, які можуть виконувати задачі незалежно, паралельно. Виконавці забезпечують випуск готових частин проєкту за певні визначені проміжки часу (спринти) [2].

Команди виконавців Scrum самокеровані, до їх обов’язків можна віднести:

- розподіл та делегування поточних задач між виконавцями;
- виконання задач та надання інформації про актуальний стан задачі;
- відстеження стану проєкту;
- визначення загальної кількості зроблених задач виконавцями за період часу, кількості задач, закінчених невчасно.

Згідно зі Scrum, розробка проєкту розбивається на декілька етапів – ітерації спринтів. Кожна ітерація починається з планування. Для кожної задачі заданий директивний строк виконання. Також попередньо задачі оцінюються і їм призначається тривалість виконання.

Необхідно спланувати спринт так, щоб мінімізувати сумарну тривалість виконання задач при нульовому запізненні.

1.1.1 Опис процесу діяльності

Зобразимо за допомогою UML-діаграми діяльності (Графічний матеріал, лист 1) процес, який ми автоматизуємо, а саме планування спринта та управління ним.

На початку власник проєкту формує проєктний резерв. На кожній новій ітерації проєкту - спринті відбувається планування. Власник проєкту разом з командою обирають задачі із резерву, які необхідно реалізувати у поточному спринті. Це пріоритетні задачі, які мають найбільш наближені до поточного дня директивні строки. Далі після спільного обговорення вони розподіляють та призначають задачі між виконавцями, таким чином вони наповнюють резерв спринта (Sprint Backlog). Після узгодження резерву задач починається спринт. Команда виконавців разом зі Scrum-майстром проводять щоденні стендапи (15-ти хвилинні зустрічі, під час яких обговорюються задачі попереднього дня, плани на поточний день, проблеми, які виникли). В процесі реалізації задач, виконавці фіксують витрачений час на задачу. Порівняння фактичного та очікуваного часу виконання додається в звіт, який потім переглядають власник проєкту та Scrum-майстер. В кінці спринта проводиться зустріч-огляд, протягом якої виконавці показують чого вони досягли протягом спринту.

1.1.2 Опис функціональної моделі

Основні ролі в системі – власник проєкту, Scrum-майстер, команда виконавців [2].

Власник проєкту володіє баченням концепту кінцевого результату розробки, тому він приймає безпосередню участь у формуванні та пріоритезації беклогу проєкту – перелік усіх задач, які необхідно реалізувати, щоб розробити проєкт.

Scrum-майстер неформальний лідер команди виконавців. Помилково казати, що Scrum-майстер розподіляє задачі між виконавцями, контролює

результати та строки. Його основний обов'язок – навчати команду принципам Scrum та впроваджувати Scrum-практики.

Команда – це люди, які будуть займатися виконанням задач. Scrum передбачає, що команди автономні. Фактично, у Scum команд відсутні керівники, адже ці команди самоорганізовані та здатні приймати рішення про свою роботу самостійно.

Далі визначимо яким чином будуть взаємодіяти актори системи між собою та з Комплексом за допомогою UML-діаграми варіантів використання (Графічний матеріал, лист 2).

Детальніше розглянемо прецеденти, зображені на діаграмі.

Для початку роботи, всі користувачі повинні авторизуватись в системі.

Власник проєкту керує проєктами, він може створювати, редагувати та видаляти проєкти. Він займається формуванням проєктної команди, а також відповідальний за створення резерву проєктних задач. Власник проєкту може переглядати графік виконання задач та проєктні звіти.

Команда виконавців займається керуванням резерву проєктних задач. З них, виконавці формують резерв спринта. Команда має можливість фіксувати поточний стан задач, та якщо задача виконана, то вказати час, витрачений на задачу. Як і Власник проєкту, команда виконавців разом зі Scrum-майстром можуть переглядати графік виконання задач та проєктні звіти.

1.2 Огляд наявних аналогів

Розглянемо детальніше вже готові рішення для систем, що мають схожі функції та підтримують розробку програмного забезпечення за технологією Scrum:

- Atlassian JIRA;
- Trello;
- HeySpace;
- Tuleap.

Проведемо аналіз недоліків та переваг наведених вище систем.

Atlassian JIRA

Jira з'явилася в 2002 році, її основною метою було дозволити програмістам відстежувати проблеми у програмному забезпеченні [3]. З часом, цей продукт почали використовувати команди не тільки з IT індустрії. Jira знайшли застосування у відстеженні різних типів проблем, завдань, робочих питань. І так вона стала одним з основних інструментів для управління проєктами. Пізніше до неї почали добудовувати різні функціональні плагіни, які розширили її можливості. Зовнішній вигляд Jira зображено на рисунку 1.1.

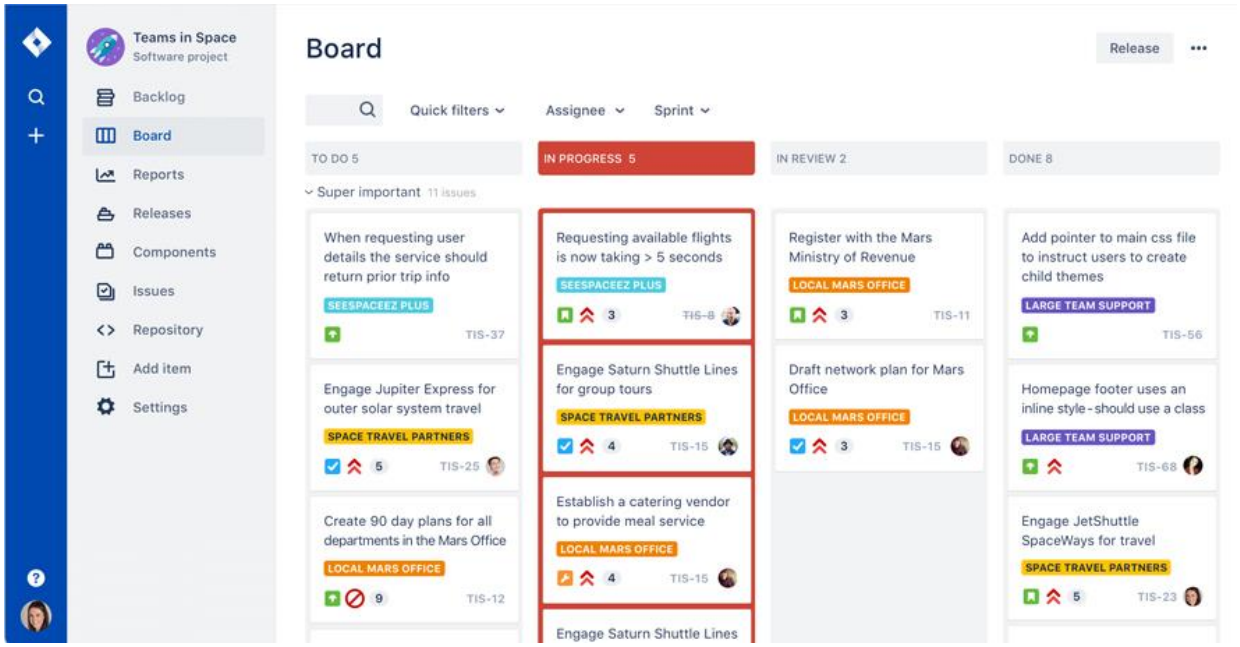


Рисунок 1.1 – Зовнішній вигляд Jira Board

Для багатьох команд зараз Jira – це робочий двигун їхніх процесів управління. Розглянемо її основні можливості.

Відстеження помилок. Це особливий тип задач, який є дуже важливим в життєвому циклі проєкта. Jira дозволяє створювати такі задачі, призначати їх на членів команди, слідкувати за змінами, які були зроблені.

Управління проєктом. Як програмне забезпечення для управління проєктами, вона дозволяє користувачам призначити необхідні завдання для завершення розробки проєкту. У Jira є можливість переглядати звіти, які

надають інформацію про директивні строки та статуси. Команда здатна співпрацювати між собою, надавати зворотній зв'язок, а також керувати схваленням запитів чи змін.

Розробка програмного забезпечення за методологією Scrum. Agile команди використовують Jira для розробки програмного забезпечення за допомогою методу Scrum.

В ході аналізу, ми виявили наступні переваги:

- Jira має інструменти для планування ітерацій роботи, яке є масштабованим, тобто можна легко пов’язати глобальні задачі проєкту з щоденними задачами команди;
- Jira можна налаштувати під свій проєкт: підтримує створення різних типів задач, звітів, форм відстеження стану проєкту;
- наявні різні типи користувачі: розробники, аналітики, тестувальники тощо;
- надає можливість інтегруватися з іншими типами систем управління проєктами.

Далі розглянемо недоліки, які були виявлені користувачами при роботі з Jira:

- перевантажений зовнішній вигляд системи, Jira має безліч функцій, але не всі з них легко знайти, фільтрації не інтуїтивно зрозуміло як застосувати;
- звіти не можна зберегти як окремий файл, лише переглядати у веб-застосунку;
- незручний мобільний застосунок.

Але, незважаючи на недоліки, Jira продовжує займати перші позиції в списку продуктів для управління проєктами.

Trello

Trello це застосунок компанії Atlassian, що з’явився в 2011 році [4]. Основна ідея Trello – організація задач у канбан-дошки. Згідно парадигми

керування проєктами канбан, задачі розділяються приблизно такі групи (в залежності від специфіки проєкту):

- резерв задач – те, що необхідно зробити;
- задачі, які вже взяті в роботу;
- задачі, які реалізовані та зараз тестуються та налагоджуються;
- задачі, які призупинені;
- задачі, які знаходяться на перевірці;
- виконані задачі.

Учасники команди розподіляються між собою задачі та поступово перетягують їх між списками. Таким чином, легко візуально відстежити на якому етапі зараз розробка проєкту. Приклад дошок Trello зображено на рисунку 1.2.

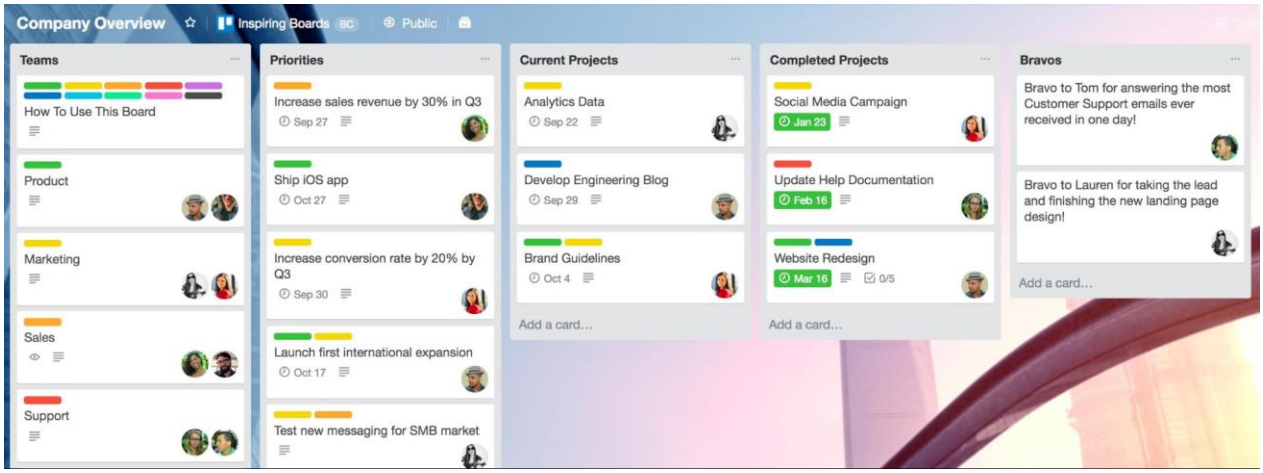


Рисунок 1.2 – Зовнішній вигляд Trello дошок

Trello можна застосовувати для вирішення різноманітних задач не тільки для управління проєктами.

Планування дня. Trello дозволяє згрупувати задачі різного типу в список, який візуально представить завантаженість вашого робочого дня.

Відслідковування часу. Trello надає можливість фіксувати директивні строки задачі; також проаналізувавши витрачений час, можна покращити свою продуктивність.

Планування відпустки. За допомогою Trello можна швидко організувати всі матеріали відпустки у списки та дошки. Така структура дозволяє тримати все перед собою та не забути важливі деталі.

Розглянемо основні переваги Trello:

- відстеження стану проєкту у реальному часі – всі оновлення одразу відображаються для інших користувачів;
- веб-застосунок має зручну навігацію, кнопки та інші елементи інтерфейсу дозволяють швидко знайти необхідну функцію;
- дуже зручний мобільний додаток;
- сповіщення – як на електронну пошту, так і push-сповіщення на телефон;
- одночасно створювати, управляти задачами може декілька користувачів;
- підтримка канбан-дошок.

До недоліків можна віднести:

- Trello підходить тільки для невеликих проєктів;
- якщо у проєкті є поділ на компоненти, під кожну таку компоненту необхідно створювати свої дошки;
- доступ тільки при підключенні до інтернету;
- не можна редагувати коментарі, які залишені під задачами.

Отже, безумовно, підтримка канбан-дошок є перевагою для scrum команд. Але в той же час, Trello не найкращий вибір для управління великими проєктами.

HeySpace

Безкоштовна онлайн-система управління проєктами HeySpace, з’явилась в 2018 році [5]. Основна ідея цього застосунку – покращити комунікацію між членами команди, що дозволяють вбудовані чат-дошки. Окрім дошок з задачами, HeySpace надає можливість створити для команди

окрему дошку для спілкування та обговорення робочих питань. Приклад зовнішнього вигляду мобільного застосунку зображено на рисунку 1.3.

Рисунок 1.3 – Зовнішній вигляд мобільної версії HeySpace

В результаті аналізу ми виявили такі переваги HeySpace:

- HeySpace є повність безкоштовним;
- підтримка канбан-дошок;
- застосунок комбінує в собі Trello (для управління проєктами) та Slack (для командної комунікації);
- повідомлення в чат-дошках можуть бути трансформовані в задачі на канбан-дошках;
- не перевантажений зовнішній вигляд, швидка навігація.

До недоліків HeySpace ми віднесли:

- відсутність пошуку між задачами-картками;
- спрощений варіант платформи обмежує її застосування для великих проєктів;
- нестабільна версія мобільного-застосунку.

Таким чином, HeySpace заохочує користувачів можливістю поєднувати комунікацію з командою та управління проєктом в одному місці. Але HeySpace підходить лише для невеликих проєктів.

Tuleap

Tuleap це платформа для правління проєктами, яка дозволяє командам використовувати різні методології розробки – Scrum, ітераційну, водоспад та інші [6]. Tuleap полегшує планування випуску програмного забезпечення, визначення пріоритетності вимог, розподіл та призначення задач на учасників команди, відстеження стану проєкту та формування звітності. У реальному часі можна відстежувати ризики, вимоги, задачі, історії змін користувачами. Зовнішній вигляд Tuleap зображено на рисунку 1.4.

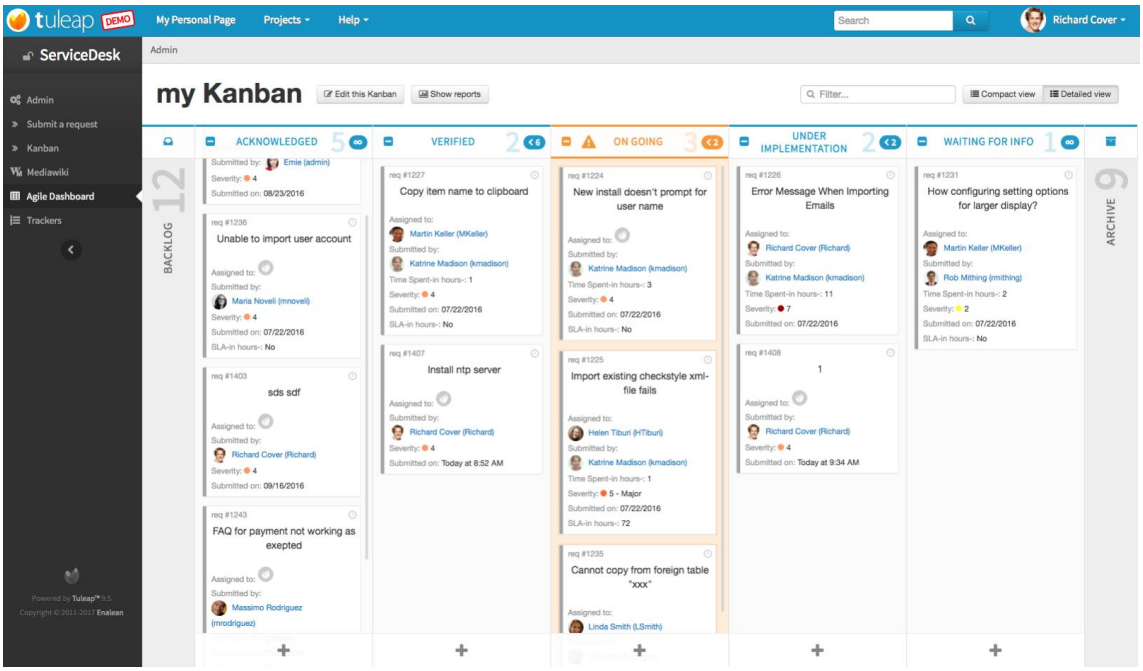


Рисунок 1.4 – Зовнішній вигляд Tuleap

В ході аналізу ми виявили наступні переваги Tuleap:

- має вбудовану систему управління ризиками;
- підтримує канбан;
- дозволяє командам керувати джерелами програмного забезпечення, використовуючи рані системи контрою версій;
- команди можуть обмінюватись проєктною документацією;
- відслідковування помилок;
- створення дошок задач на льоту;
- можливість створювати звіти;

Серед недоліків можна виділити:

- не так часто застосовується у комерційних проєктах, порівняно з Jira;
- не має можливості інтеграції з Slack;
- не має можливості інтеграції з HipChat.

Отже, Tuleap має широкий набір функцій та можливостей. Як видно з наведених переваг, Tuleap можна застосовувати для управління великими та маленькими проєктами.

Відмінності комплексу задач від перелічених аналогів

Основна особливість комплексу – автоматичний розподіл задач між учасниками виробництва. Також комплекс дозволятиме команді відстежувати поточний стан задач, визначати загальну кількість зроблених задач виконавцями за період часу, кількість задач, закінчених невчасно.

1.3 Постановка задачі

1.3.1 Призначення розробки

Комплекс задач призначений для підтримки процесу управління командою виконавців, а саме:

- для розподілу задач між учасниками виробництва;
- для відстеження поточного стану задач;
- для визначення загальної кількості зроблених задач виконавцями за період часу, кількості задач, закінчених невчасно;
- для управління задачею виконавцями: зміна статусу задачі, у відповідності з її поточним станом.

1.3.2 Цілі та задачі розробки

Метою даного комплексу задач є підвищення ефективності роботи команди виконавців, за рахунок максимального дотримання директивних строків та мінімізації середнього часу виконання робіт.

Для досягнення поставленої мети необхідно реалізувати такі задачі:

- ведення виконавців;
- ведення проєктів;
- ведення задач;
- ведення проєктних команд;
- ведення задач виконавців;
- складання плану робіт виконавцями;
- формування звітності.

Комплекс задач може застосовуватись як окрема система планування роботи команди так і плагін до відомих систем командної розробки (наприклад, Atlassian Jira).

Висновок до розділу

Виконано проєктування комплексу задач з підтримки процесу управління командою виконавців; описане предметне середовище; розглянуто ролі користувачів та процеси їхньої взаємодії; проаналізовані існуючі аналоги систем, виявлено їх переваги та недоліки; розроблені діаграма діяльності та діаграма варіантів використання; поставлені цілі, сформована мета, визначено функції, які необхідно реалізувати.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Для комплексу задач вхідними даними є такі документи:

а) проєктний резерв задач;

б) резерв задач спринта;

в) проєктна команда виконавців.

Опишемо документи детальніше в таблиці 2.1.

Таблиця 2.1 – Реквізити документів вхідних даних

№	Найменування	Реквізити	Опис документу
1	Проєктний резерв задач	Назва задачі; Номер задачі; Директивний строк; Відносна оцінка; Оцінка часу виконання; Опис	Містить перелік задач, які ще не взяті в реалізацію та не призначені виконавцям.
2	Резерв задач спринта	Назва задачі; Номер задачі; Директивний строк; Відносна оцінка; Оцінка часу виконання; Опис; Статус	Містить перелік задач, які ще не взяті в реалізацію та не призначені виконавцям, але вже обрані в поточний спринт.
3	Проєктна команда виконавців	Ім'я; Посада; Електронна пошта	Користувачі, які додані на проєкт, та мають можливість виконувати проєктні задачі.

2.2 Вихідні дані

Для комплексу задач вихідними даними є такі документи:

а) календарний план виконання задач (розклад виповнення задач для кожного виконавця);

б) календарний план виконання задач (розклад виповнення задач для команди);

в) звіт про проведення спринта.

Опишемо документи детальніше в таблиці 2.2.

Таблиця 2.2 – Реквізити документів вхідних даних

№	Найменування	Реквізити	Опис документу
1	Календарний план виконання задач для кожного виконавця	Назва задачі; номер задачі; директивний строк; відносна оцінка; оцінка часу виконання; опис; ім’я виконавця; фактичний час виконання; час завершення виконання	Розклад виповнення задач для кожного виконавця, визначений порядок виконання.
2	Календарний план виконання задач для команди	Назва задачі; номер задачі; директивний строк; відносна оцінка; оцінка часу виконання; опис; статус; ім’я виконавця; фактичний час виконання; час завершення виконання	Розклад виповнення задач для команди, визначений порядок виконання задач для команди в цілому.
3	Звіт про проведення спринта	Фактичний час виконання задач; кількість задач, завершені кожним виконавцем; кількість задач, які виконані невчасно; сумарна кількість відносних оцінок за виконані задачі впродовж спринта	Графічне представлення результатів, аналітичне.

Усі звіти видаються на екран. Приклади звітів на екранних формах наведені в графічних матеріалах, лист 3.

2.3 Опис структури бази даних

Наведемо перелік таблиці та поля у таблиці 2.3.

Таблиця 2.3 – Перелік таблиць та полів

№	Назва таблиці	Поля
1	Задачі	Id задачі; оцінка часу виконання; дата початку роботи над задачею; дата завершення роботи над задачею; статус; id користувача (зовнішній ключ), який виконав/виконуватиме задачу; id спринта (зовнішній ключ), до якого прив’язана задача; id проєкту (зовнішній ключ); директивний строк; відносна оцінка; опис; назва; код
2	Спринти	Id спринта; назва; id проєкту (зовнішній ключ); дата початку спринта; дата завершення спринта
3	Проєкти	Id проєкту; код; назва; id власника проєкту (зовнішній ключ); id поточного спринта (зовнішній ключ); директивний строк; дата початку проєкту

Продовження таблиці 2.3

№	Назва таблиці	Поля
4	Системні користувачі	Id користувача; повне ім’я; пароль; логін; посада; електронна пошта
5	Системні групи	Id групи; назва
6	Групи - Користувачі	Id групи (зовнішній ключ); id користувача (зовнішній ключ)
7	Користувачі - Проєкти	Id користувача (зовнішній ключ); id проєкту (зовнішній ключ)
8	Сесії	Id сесії; id користувача (зовнішній ключ); токен; ip; дата

Схема бази даних наведена у графічних матеріалах, лист 4. Розглянемо детальніше кожну з наведених таблиць.

Таблиця «Задачі» містить в собі перелік всіх наявних задач системи. Опис полів наведений у таблиці 2.4.

Таблиця 2.4 – Опис полів таблиці «Задачі»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id задачі	Number (Число)	Унікальний ідентифікатор задачі в системі	Id
2	Оцінка часу виконання	Number (Число)	Приблизний час виконання задачі, який виставляється заздалегідь командою розробки	Estimate

Продовження таблиці 2.4

№	Назва поля	Тип поля	Опис	Назва поля в БД
3	Дата початку роботи над задачею	DateTime (Дата з часом)	Дата, коли задача була переведена в статус «In Progress»	Date_from
4	Дата завершення роботи над задачею	DateTime (Дата з часом)	Дата, коли задача була переведена в статус «Completed» (Завершено) або «Canceled» (Відхилено)	Date_to
5	Статус	Enumeration (Перелік)	Стан задачі. Можливі значення: - Open (Відкрита); - In Progress (В обробці); - Canceled (Відхилена); - Completed (Завершена).	Status
6	Id користувача (зовнішній ключ), який виконав/виконуватиме задачу	Number (Число)	Унікальний ідентифікатор користувача, за яким закріплена задача.	Used_id
7	Id спринта (зовнішній ключ), до якого прив’язана задача	Number (Число)	Унікальний ідентифікатор спринта, за яким закріплена задача.	Sprint_id
8	Id проєкту (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор проєкту, за яким закріплена задача.	Project_id
9	Директивний строк	DateTime (Дата з часом)	Строк, до якого задача повинна бути завершена.	Due_date

Продовження таблиці 2.4

№	Назва поля	Тип поля	Опис	Назва поля в БД
10	Відносна оцінка	Number (Число)	Відносна оцінка ваги задачі – показник у Scrum.	Story_ points
11	Опис	String (Рядковий тип даних)	Послідовність дій для виконання задачі/вхідні дані/будь-яка інформація, яка може бути використана для виконання задачі.	Description
12	Назва	String (Рядковий тип даних)	Короткий опис задачі.	Name
13	Код	String (рядковий тип даних)	Проєктний ідентифікатор задачі, складається з проєктного коду та номеру задачі.	Code

Таблиця «Спринти» містить в собі перелік всіх наявних спринтів системи. Опис полів наведений у таблиці 2.5.

Таблиця 2.5 – Опис полів таблиці «Спринти»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id спринта	Number (Число)	Унікальний ідентифікатор спринта в системі	Id
2	Назва	String (Рядковий тип даних)	Короткий опис спринту.	Name
3	Id проєкту (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор спринту.	Project_id
4	Дата початку спринта	DateTime (Дата з часом)	Дата, коли спринт став активним.	Start_date
5	Дата завершення спринта	DateTime (Дата з часом)	Дата, коли спринт було завршено.	End_date

Таблиця «Проєкти» містить в собі перелік всіх наявних проєктів системи. Опис полів наведений у таблиці 2.6.

Таблиця 2.6 – Опис полів таблиці «Проєкти»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id проєкту	Number (Число)	Унікальний ідентифікатор спринта в системі	Id
2	Код	String (Рядковий тип даних)	Ідентифікатор проєкту, що складається з перших літер назви проєкту.	Code
3	Назва	String (Рядковий тип даних)	Короткий опис проєкту.	Name
4	Id власника проєкту (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор користувача, який створив проєкт.	Owner_id
5	Id поточного спринта (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор активного спринта.	Active_sprint_id
6	Директивний строк	DateTime (Дата з часом)	Строк, до якого проєкт повинен бути завершений.	Due_date
7	Дата початку проєкту	DateTime (Дата з часом)	Дата, коли проєкт було створено.	Start_date

Таблиця «Системні користувачі» містить в собі перелік всіх наявних користувачів системи. Опис полів наведений у таблиці 2.7.

Таблиця 2.7 – Опис полів таблиці «Системні користувачі»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id користувача	Number (Число)	Унікальний ідентифікатор користувача в системі.	Id
2	Повне ім’я	String (Рядковий тип даних)	Ім’я та прізвище користувача.	Full_name
3	Пароль	String (Рядковий тип даних)	Пароль для входу користувача в систему.	Password
4	Логін	String (Рядковий тип даних)	Логін для входу користувача в систему.	Login
5	Посада	String (Рядковий тип даних)	Посада, яку займає користувач в компанії.	Position
6	Електронна пошта	String (Рядковий тип даних)	Електронна пошта користувача.	Email

Таблиця «Системні групи» містить в собі перелік всіх наявних груп користувачів системи. Опис полів наведений у таблиці 2.8.

Таблиця 2.8 – Опис полів таблиці «Системні користувачі»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id групи	Number (Число)	Унікальний ідентифікатор групи в системі.	Id
2	Назва	String (Рядковий тип даних)	Назва користувацької групи	Name

Таблиця «Групи-Користувачі» відображає зв’язок багато до багатьох між таблицями Користувачів та Груп. Опис полів наведений у таблиці 2.9.

Таблиця 2.9 – Опис полів таблиці «Групи-Користувачі»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id групи (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор групи в системі.	Group_id
2	Id користувача (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор користувача в системі.	User_id

Таблиця «Користувачі-Проєкти» відображає зв’язок багато до багатьох між таблицями Користувачів та Проєкти. Опис полів наведений у таблиці 2.10.

Таблиця 2.10 – Опис полів таблиці «Користувачі-Проєкти»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id користувача (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор користувача в системі.	User_id
2	Id проєкту (зовнішній ключ)	Number (Число)	Унікальний ідентифікатор проєкту в системі.	Project_id

Таблиця «Сесії» системна таблиця для авторизації користувачів в системі. Опис полів наведений у таблиці 2.11.

Таблиця 2.11 – Опис полів таблиці «Сесії»

№	Назва поля	Тип поля	Опис	Назва поля в БД
1	Id сесії	Number (Число)	Унікальний ідентифікатор сесії користувача.	Id
2	Id користувача (зовнішній ключ);	Number (Число)	Унікальний ідентифікатор користувача в системі.	User_id

Продовження таблиці 2.11

№	Назва поля	Тип поля	Опис	Назва поля в БД
3	Токен	String (Рядковий тип даних)	Унікальний рядок, який допомагає системі розпізнавати користувача.	Token
4	IP	String (Рядковий тип даних)	IP адреса користувача.	Ip
5	Дата	String (Рядковий тип даних)	Додаткова інформація про сесію.	Data

Висновок до розділу

У даному розділі вказані вхідні та вихідні документи для комплексу задач. Виконано проєктування бази даних, з детальним описом таблиць, полів та зв’язків між таблицями.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Для комплексу, який розробляється в рамках даного дипломного проєкту, ставиться математична задача складання розкладу виконання задач з різними директивними строками командою виконавців. Розглянемо детальніше цю задачу, оскільки для її розв'язання розроблено наближений алгоритм, який можна використовувати для близьких задач.

3.1 Змістовна постановка задачі

Існує деяка компанія, яка отримує замовлення на проєкти. На кожен проєкт компанія призначає команду виконавців (Scrum Team). Команди складаються з множини учасників, які можуть виконувати задачі незалежно, паралельно. Згідно зі Scrum, розробка проєкту розбивається на декілька етапів – ітерації спринтів. Кожна ітерація починається з планування. Для кожної задачі заданий директивний строк виконання. Також попередньо задачі оцінюються і їм призначається тривалість виконання.

Необхідно спланувати спринт так, щоб мінімізувати сумарну тривалість виконання задач при нульовому запізненні.

Оскільки використовуємо математичний апарат теорії розкладів, то далі вважатимемо, що задачі – це роботи, а виконавці – це машини.

3.2 Математична постановка задачі

Нехай маємо m машин, які є ідентичними, або пропорційними. Вони повинні виконати n робіт, які мають такі характеристики [1]:

- p_j – тривалість виконання роботи;
- d_j – директивний строк виконання роботи;
- C_j – момент часу, коли робота j завершує своє виконання.

Розглянемо характеристики розкладу [1]:

- C_{max} – момент часу, коли завершилось виконання останньої роботи в системі;
- T_j – час запізнення роботи j.

Необхідно скласти такий розклад, для кого буде виконуватись:

- усі роботи повинні виконатись без запізнення, значення максимального запізнення T_{max} повинно дорівнювати нулю;
- мінімальний сумарний час завершення виконання робіт.

Згідно нотації Грехема, маємо задачу, яку можна описати формулою 3.1 [1]:

$$Pm|d_j|\, max\, T_j = 0\ , \sum C_j, \tag{3.1}$$

де Pm – m ідентичних машин;

d_j – директивні строки робіт;

$max\, T_j = 0$ – значення максимального запізнення T_{max} повинно дорівнювати 0;

$\sum C_j$ – мінімальний сумарний час завершення виконання робіт.

3.3 Обґрунтування методу розв'язання

Для поставленої задачі наразі немає розроблених алгоритмів вирішення. Далі буде запропонований алгоритм побудови розкладу, який створений на основі результатів близьких задач, а саме:

а) задача складання розкладу для однієї машини, та n робіт з різними директивними строками, який оптимальний за двома критеріями: нульове максимальне запізнення, мінімізація середньої тривалості виконання робіт (далі задача $1|d_j|\, max\, T_j = 0\ , \sum C_j$);

б) задача складання розкладу для m ідентичних машин, та n робіт, в якому мінімізується максимальний час завершення виконання робіт (далі задача $Pm||C_{max}$).

Розглянемо детальніше алгоритми цих задач.

3.3.1 Задача $1|d_j|\,max\,T_j = 0\ ,\sum C_j$

Алгоритм побудови розкладу базується на теоремі Джексона (Правило найбільш раннього директивного строку) та теоремі Сміта.

Теорема Джексона (Правило найбільш раннього директивного строку Earliest Due Date). Коли машина завершує виконання останньої призначеної роботи, наступною їй призначається робота з найбільш раннім директивним строком [7][9].

При впорядкування робіт за даним правилом, отримуємо розклад, для якого значення максимального запізнення дорівнює нулю. Для даної задачі можливо отримати також інший розклад, за якого роботи також не запізнюються, але впорядковані за директивними строками краще за іншими критеріями. Якщо за додатковий критерій обрати мінімізацію середньої тривалості проходження, то необхідно використовувати наступну теорему.

Теорема Сміта [1]. Якщо для системи $n|1$ існує таке впорядкування, що максимальне запізнювання робіт дорівнює 0, то існує і впорядкування з роботою K в останній позиції, яке зберігає нульове максимальне запізнювання і одночасно мінімізує середню тривалість проходження тоді і тільки тоді, коли:

а) $d_K \geq \sum_{i=1}^{n} p_i$ (робота виконується останньою, якщо це не призводить до її запізнювання),

б) $p_K \geq p_i$ для всіх i таких, що $d_i \geq \sum_{j=1}^{n} p_j$ (її тривалість максимальна серед всіх робіт, виконання яких в останню чергу приводить до запізнювання).[8][10]

Алгоритм А1 побудови оптимального розкладу для задачі $1|\mathrm{d_j}|\max \mathrm{T_j} = 0\ ,\sum \mathrm{C_j}$ наведений на рисунку 3.1.

Рисунок 3.1 – Блок-схема алгоритму А1

3.3.2 Задача $Pm||C_{max}$

Задача складання розкладу для m>1 кількості ідентичних машин. Кожна робота може бути виконана будь-якою машиною.

Алгоритм А2 побудови оптимального розкладу для задачі $Pm||C_{max}$ наведений на рисунку 3.2.

Рисунок 3.2 – Блок-схема алгоритму А2

Даний розклад можна покращити з огляду на максимальний час завершення робіт, попарно переставляючи роботи одного рівня між машинами.

3.4 Опис методів розв'язання

Опишемо попередні умови до алгоритму.

Нехай теоретично мінімальний час, за який усі машини змогли б завершити всі роботи обчислюється як сума тривалостей всіх робіт поділена на кількість машин, і це значення округлене до верхнього цілого, формула 3.2 [11].

$$C^* = \left\lceil \frac{\sum p}{m} \right\rceil. \quad (3.2)$$

Необхідна, але не достатня, умова того, що можливо побудувати допустимий розклад (без запізнень): усі роботи повинні мати такі директивні строки, які будуть більшими за теоретично мінімальний час: $\forall\, d_j > C^*$ [1].

Алгоритм побудови допустимого розкладу базується на ідеях алгоритмів *A1* та *A2*, описаний у пунктах 3.2.1, 3.2.2.

Алгоритм А3 побудови допустимого розкладу для задачі $Pm|d_j|maxT_j = 0, \sum C_j$ наведений на рисунку 3.3.

Проілюструємо результат роботи алгоритму на діаграмі Гантта. Нехай маємо такі вхідні параметри:

- 5 паралельних машин;
- 50 робіт, для яких випадковим чином згенеровані значення тривалостей виконання та директивних строків.

Діаграма зображена на рисунку 3.4. Горизонтальна числова пряма відображає часові проміжки. Для кожної машини наведена послідовність робіт у вигляді прямокутників з порядковим номером роботи та тривалістю виконання роботи.

Рисунок 3.3 – Блок-схема алгоритму А3

Рисунок 3.4 – Діаграма Гантта для отриманого розкладу

3.4.1 Дослідження запропонованого алгоритму

Дослідимо умову, за якої запропонований алгоритм не буде давати допустимий розклад. Ми вже визначили необхідну умову раніше: всі роботи повинні мати такі директивні строки, які більші за теоретично мінімальний час – $\forall\, d_j > C^*$.

Дослідимо зміни в розкладі при зменшенні директивних строків. Нехай для фіксованої кількості машин (5) згенеруємо роботи (50) з нежорсткими директивним строками. Далі складемо розклад за описаним алгоритмом А3.

Поступово будемо змінювати значення директивних строків, зменшуючи їх з кожною ітерацією на 1, та перебудовувати розклад, доки не з’являться запізнення робіт.

Коли значення максимуму запізнення стає більше нуля, обчислюємо відношення суми директивних строків до суми тривалостей робіт: $\sum d_j \,/\, \sum p_j$.

Після цього для поточної кількості робіт та даного набору значень директивних строків завершуємо ітерацію експерименту.

Повторимо експеримент, збільшуючи кількість робіт та генеруючи різні набори значень директивних строків. Для кожного нового набору виконати 200 разів.

Розглянемо детальніше мінімальні значення директивних строків, за яких ще немає запізнень, тобто для яких наш алгоритм зможе дати допустимий розклад. Отримали результати, наведені в таблиці 3.1. Залежність степені жорсткості директивних строків від кількості робіт графіком зображена на рис. 3.5.

Таблиця 3.1 – Результати дослідження [1]

Кількість робіт	**Середній min директивний термін**	C^*	$\mathbf{C_{max}}$	**Степінь жорсткості директивних строків** $\sum \boldsymbol{d_j} / \sum \boldsymbol{p_j}$.
200	256,92	257,375	258,245	12,601
250	317,94	318,370	318,940	15,143
300	381,23	381,700	382,200	17,652
350	444,54	444,920	445,260	20,146
400	508,24	508,715	508,980	22,628
450	573,05	573,540	573,785	25,121
500	638,58	638,970	639,195	27,625
550	702,62	703,070	703,225	30,120
600	764,36	764,790	764,855	32,573
650	826,86	827,295	827,350	35,096
700	891,73	892,245	892,265	37,595
750	955,57	955,995	956,030	40,117
800	1018,80	1019,310	1019,315	42,633
850	1085,40	1085,900	1085,910	45,089
900	1145,97	1146,405	1146,410	47,612
950	1211,09	1211,540	1211,555	50,130
1000	1275,68	1276,060	1276,060	52,623

Рисунок 3.5– Графік залежності степені жорсткості директивних термінів від кількості робіт

Отже, нами запропонована статистично обґрунтована метрика, яка дозволяє з достатньою впевненістю визначити, чи можливо побудувати двокритеріальний розклад для заданої множини робіт – степінь жорсткості директивних строків [1].

Висновок до розділу

У даному розділі було виконано огляд двох існуючих алгоритмів складання розкладів:

- алгоритм для складання розкладу для однієї машини, та n робіт з різними директивними строками, за двома критеріями: нульове максимальне запізнювання, мінімізація середньої тривалості виконання робіт;
- алгоритм для складання розкладу для m ідентичних машин, та n робіт, який мінімізує максимальний час завершення виконання робіт.

Було запропоновано наближений алгоритм для складання розкладу виконання робіт з різними директивними строками ідентичними машинами за двома критеріями. В результаті дослідження алгоритму було досліджено умови, за яких можна отримати допустимий розклад без запізнень, для якого було запропоновано показник жорсткості директивних строків.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

При створенні комплексу використовувались наступні технології:

- PostgreSQL – система управління базами даних;
- JavaScript – мова програмування;
- Angular – фреймворк мови програмування JavaScript для розробки веб-застосувань;
- Node.js – програмна платформа JavaScript;
- Node.js Starter Kit [12];
- Angular Material – база компонентів для зовнішнього вигляду екрану[13];
- Akita State Management – бібліотека для Angular [14].

Розглянемо детальніше технології PostgreSQL, JavaScript, Angular, Node.js та обґрунтуємо їх застосування для розробки комплексу.

В додатку Б наведений код основних компонентів програми.

4.1.1 PostgreSQL

PostgreSQL об’єктно-реляційна система управління базами даних, яку часто використовують для веб баз даних. Це одна з найперших систем, яка дозволила користувачам керувати як структурованими, так і неструктурованими даними [15].

Вибір PostgreSQL як системи управління базами даних в Комплексі зумовлено такими перевагами:

- працює з великою різноманітністю форматів даних, включаючи деякі досить зручні типи, такі як 'масив' (англ. ‘array’), який може забезпечити швидке, спеціальне агрегування в результатах запитів;
- швидке та стабільне виконання запитів;

- є платформою з відкритим кодом, наявна велика кількість активних спільнот, які активно підтримують своїх користувачів;
- двигун управління PostgreSQL є масштабованим, що дає можливість обробляти терабайти даних.

Недоліком є те, що при опрацюванні дійсно великих об’ємів даних, може спостерігатись певне сповільнення в роботі. Але ця проблема на даному етапі розробки не виникне.

Слід зазначити, що функціональність PostgreSQL не обмежується лише наданням допомоги розробникам, але й адміністраторам, щоб захистити цілісність даних та забезпечити таке середовище бази даних, в якому можна легко керувати своїми даними, незалежно від того, наскільки великий чи малий набір даних. На додаток до цього, він безкоштовний і масштабований, а також є можливість визначати власні типи даних або формувати власні функції без шкоди для бази даних.

4.1.2 JavaScript

JavaScript є дуже потужною мовою скриптів на стороні клієнта. JavaScript дозволяє створювати динамічний контент для Інтернету. JavaScript - це мова з відкритим кодом, є крос-платформною. Він не вимагає компіляції та інтерпретується об'єктно-орієнтованими можливостями. Також він працює з різними іншими мовами програмування [16]. І це є причиною його широкого використання у всьому світі. Багато популярних веб-сайтів та веб-додатків, таких як Google, Amazon, PayPal тощо використовують цю мову. Розширення файлу JavaScript-файлу .js.

Більшість додатків працюють завдяки взаємодії між клієнтом (пристроєм користувача) та віддаленим сервером. Клієнт надсилає запит на дані з сервера. Сервер отримує запит, обробляє його, а потім відповідає. Відповідь, що надсилається назад, є у зрозумілому користувачу форматі. Але

цей процес вимагає часу, а також ресурсів. Хоча нам зазвичай і потрібне це з'єднання, у деяких проектах JavaScript часто допомагає уникнути цього.

JavaScript дозволяє валідувати форми без вхідних даних від сервера, зменшивши трафік. Він пропонує чудові інструменти для більш інтерактивного та зручного веб-дизайну. Деякі основні функції JavaScript [17]:

- автозаповнення: у вікні пошуку подаються пропозиції, виходячи з того, що користувач уже ввів;
- перевірка форми: якщо користувачі роблять помилку під час заповнення форми, JavaScript негайно повідомляє їх про помилку, уникаючи заповнення її знову;
- виправляє проблеми з компонуванням, щоб уникнути перекриття елементів на сторінці;
- додає анімацію на сторінку, щоб зробити її привабливішою.

Нижче наведено основні можливості JavaScript:

- підтримує об'єктно-орієнтовані концепції програмування;
- незалежний від платформи та чутливий до регістру;
- надає різні вбудовані функції, такі як alert(), prompt() тощо;
- можливість обробки винятків;
- дозволяє використовувати функції з іменем або без нього.

Саме JavaScript обрано для розробки комплексу, оскільки в цій мові є гнучка і динамічна система типів, що дозволяє легко будувати потрібні абстракції та прототипувати предметну область швидко, а також підходить для сучасних тенденцій – писати асинхронний та реактивний код.

4.1.3 Angular

Angular – базований на TypeScript відкритий фреймворк для веб-застосувань.

Angular відмінно підходить для створення додатків для фронтенду. Якщо ви розробляєте новий додаток з нуля, ви можете використовувати його та розгортати його, навіть якщо на вашому сервері немає NodeJS. Більшість ресурсів рекомендують використовувати Angular для великих проєктів і для великих команд, оскільки він надає всі інструменти для вирішення складних завдань і обробки великої кількості даних [18].

До переваг Angular можна віднести:

- відмінно справляється із підтримкою веб-компонентів: пропонує більш прості API та вкладені компоненти, дбає про ледаче завантаження та багато іншого, що стало стандартом сучасного Інтернету;
- підходить для масштабних проєктів, що значно збільшує можливості його використання;
- має просту систему налагодження, що забезпечує ідеальну основу для стабільності продуктів.

Angular обрано для розробки комплексу оскільки він є найбільш насичений готовим функціоналом фреймоворк, який дозволяє одразу починати писати свої компоненти та сервіси, не реалізовуючи системні функції.

4.1.4 Node.js

Node.js заснований на JavaScript і найчастіше використовується для бекенд (англ. backend) і фронтенд (англ. frontend) розробки. Node.js дозволяє розробникам використовувати JavaScript через стек, застосовуючи технологічно подібні рішення до кожної проблеми в проєкті. Звичайно, ми не можемо вирішити всі проблеми одним рішенням, але використання подібних мов програмування для різних сегментів коду полегшує тестування та обслуговування. Це головна перевага впровадження Node.js в проєкт розробки програмного забезпечення. Використання Node.js разом з JavaScript

для серверів має сенс - таким чином, весь код побудований з однаковими принципами [19].

Використовуючи Node.js для бекенду, автоматично маємо такі переваги повноцінної розробки JavaScript стека, такі як:

- підвищення ефективності та загальної продуктивності розробника;
- обмін кодом та повторне використання;
- швидкість та продуктивність;
- легкий обмін знаннями в команді;
- величезна кількість безкоштовних інструментів.

Node.js обрано для реалізації комплексу через те, що зручно використовувати одну мову для розробки фронтенду та бекенду, а також підхід виклику віддалених процедур (англ. Remote procedure call, RPC) і одна мова програмування дозволяють викликати процедури на сервері так, ніби це одна програма.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Комплекс рекомендовано використовувати на комп’ютері, що має такі характеристики:

- операційна система – Windows 7/8/10;
- процесор – не нижче Intel Pentium CPU N3700 1.6 ГГц;
- оперативна пам’ять – 1 Гб;
- вільне місце жорсткого диску – 5 Гб.

4.3 Архітектура програмного забезпечення

4.3.1 Архітектурна діаграма

Архітектурна діаграма наведена у графічних матеріалах, лист 4. Розглянемо детальніше елементи діаграми.

У програми є Головний модуль (англ. application.js), це клас, який є головним контекстом управління, тобто він керує всіма іншими модулями. Він завантажує статичну інформацію (config.js), як конфігурація серверу (config/server.js), конфігурація бази даних (database.js) та SSL сертифікати (cert). Також Головний модуль сканує папку з файлами з API-функціями і кешує функції, розміщуючи їх в «пісочниці».

На основі імен API-файлів створюються кінцеві точки (англ. endpoint) POST модулем Сервер (server.js). Сервер слухає кінцеві точки на заданому порті й перевіряє авторизацію під час кожного запиту.

Модуль авторизації (auth.js) відповідає за хешування паролів, створення сесій, перевірку сесій, створення користувачів.

Шар доступу до даних (db.js), або модуль доступу по даних створює пул з'єднань з базою даних та надає іншим модулям інтерфейс взаємодії з даними, інкапсулюючи в собі логіку створення запитів.

4.3.2 Діаграма послідовності

Діаграма послідовності наведена у графічних матеріалах, лист 5. Розглянемо детальніше процеси на діаграмі.

Є два основних сценарії. Перший – це логін в системі користувача. Користувач відправляє свої логін та пароль на сервер, далі сервер знаходить користувача в базі даних за логіном, хешує введений пароль та порівнює з хешем у базі. Якщо паролі співпадають сервер створює сесію в базі та повертає Id користувача та токен. В іншому випадку повертається помилка і користувач залишається на сторінці логіну.

Другий сценарій стосується всіх інших запитів. З кожним запитом користувач надсилає токен з браузерних cookie. Сервер шукає активну сесію по токену. Якщо такої немає – повертає помилку і користувач направляється в вікно логіну. Якщо така сесія знайдена – запит обробляється відповідною

закешованою API-функцією. Відбуваються маніпуляції з даними і результат повертається користувачу.

4.3.3 Схема структурна взаємодії клієнта, сервера та бази даних

На рисунку 4.1 зображена схема структурна взаємодії. Розглянемо більш детально її елементи та зв'язки між ними.

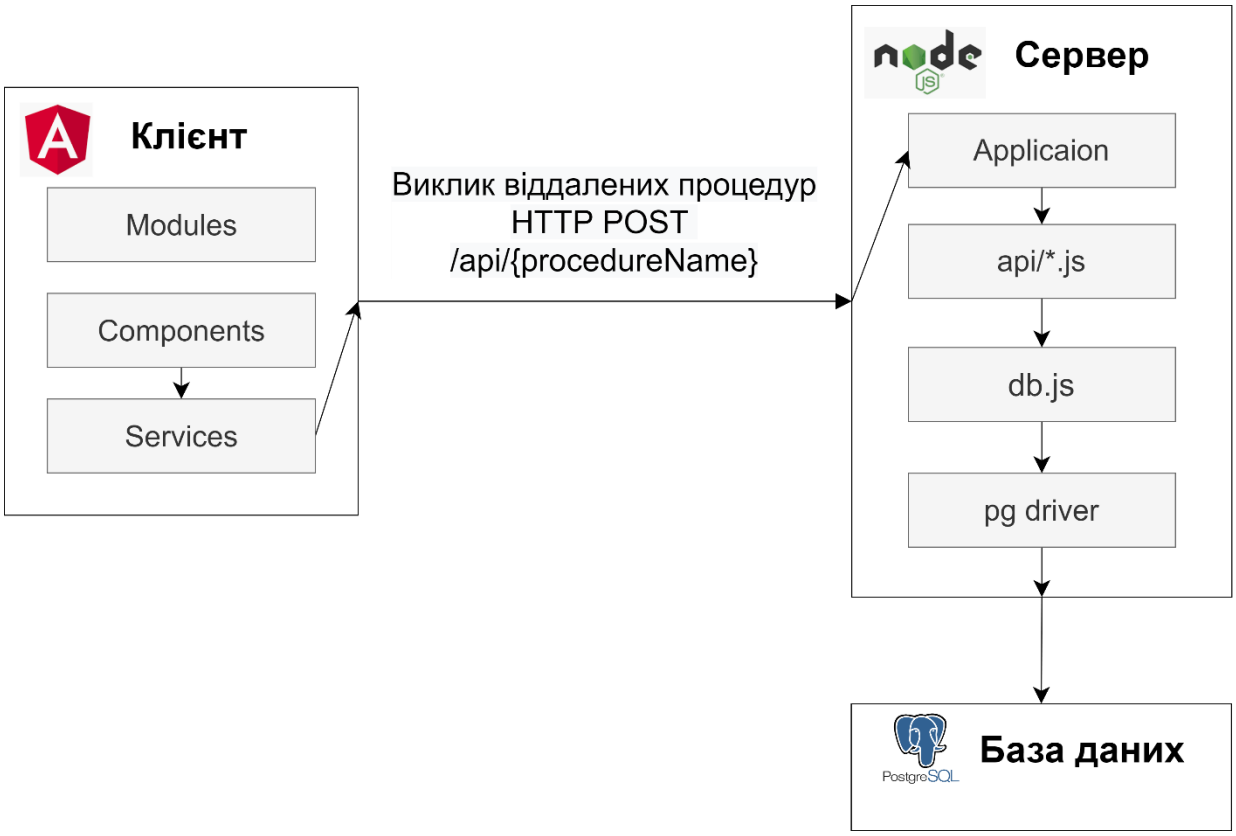


Рисунок 4.1 – Схема структурна взаємодії

Браузер та сервер взаємодіють за концепцією «Виклик віддалених процедур». Суть концепції полягає в тому, що коли комп'ютерна програма викликає процедуру на виконання в іншому адресному просторі (наприклад, на іншому комп'ютері в спільній мережі), процедура виконується так, як ніби викликана локально. Це форма взаємодії клієнт-сервер, що зазвичай реалізується через систему передачі повідомлень запит-відповідь.

Браузер надсилає HTTP POST запит на IP-адресу, на якій запущено сервер Node.js. Сервер реагує на запит: може надіслати новий запит в базу даних, обробити дані, оновити їх в базі, повернути дані користувачу. Імена

кінцевих точок (англ. endpoint) співпадають з назвами віддалених процедур (procedureName), а тіло запиту містить в собі параметри процедури. На початку роботи сервер відкриває певну обмежену кількість зв'язків та тримає їх відкритими, щоб не створювати новий зв'язок на кожен запит. Коли запити приходять на сервер, він використовує ці зв'язки.

4.3.4 Специфікація функцій

У таблиці 4.1 наведено опис основних інтерфейсів.

Таблиця 4.1 – Опис основних прикладних програмних інтерфейсів (англ. Application Programming Interface, API)

Назва інтерфейсу	Вхідні параметри	Параметри відповіді	Опис інтерфейсу
assignTaskToSprint	sprintId: number; taskId: number	response: string	Призначає задачу до спринта, надсилає у відповідь чи успішна дія
assignUser	userId: number; projectId: number	response: string	Призначає користувача на проєкт, надсилає у відповідь чи успішна дія
createSprint	export type Sprint = Readonly<{ id?: number; name: string; projectId: number; startDate?: string; endDate?: string; readonly?: boolean; }>;	response: string	Створює спринт, надсилає у відповідь чи успішна дія
usersNotInProject	projectId: number;	userId: number;	Шукає користувачів, які не призначені на проєкт, повертає масив користувачів

Продовження таблиці 4.1

Назва інтерфейсу	Вхідні параметри	Параметри відповіді	Опис інтерфейсу
createProject	export type Project = Readonly<{ id?: number; code: string; name: string; ownerId: number; activeSprintId?: number; dueDate?: string; startDate?: string; sprint?: string; userCount?: number; backlogCount?: number; storyPoints?: number; overdues?: number; sprintCode?: string; }>;	response: string	Створює проєкт, надсилає у відповідь чи успішна дія
createSprintSchedule	sprintId: number;	response: string	Створює розлад для спринта, надсилає у відповідь чи успішна дія
editProject	export type Task = Readonly<{}>;	response: string	Редагує проєкт, надсилає у відповідь чи успішна дія

Продовження таблиці 4.1

Назва інтерфейсу	Вхідні параметри	Параметри відповіді	Опис інтерфейсу
createTask	export type Task = Readonly<{ id?: number; estimate?: number; dateFrom?: string; dateTo?: string; status: TaskStatus; userId?: number; sprintId?: number; projectId: number; dueDate?: string; storyPoints?: number; description?: string; name: string; code: string; order?: number; color?: string; userName?: string; }>;	response: string	Створює задачу, надсилає у відповідь чи успішна дія
editSprint	export type Sprint = Readonly<{}>;	response: string	Редагує спринт, надсилає у відповідь чи успішна дія
editTask	export type Task = Readonly<{}>;	export type Task = Readonly<{}>;	Редагує задачу, надсилає у відповідь оновлену задачу
deleteProject	projectId: number	response: string	Видаляє проєкт, надсилає у відповідь чи успішна дія

Продовження таблиці 4.1

Назва інтерфейсу	Вхідні параметри	Параметри відповіді	Опис інтерфейсу
getProjectBacklog Tasks	sprintId: number; projectId: number;	export type Task = Readonly<{}> ;	Отримує проєктний резерв, надсилає у відповідь масив задача
getTasksByProject	projectId: number;	export type Task = Readonly<{}> ;	Отримує проєктні задачі, надсилає у відповідь масив задач
registerUser	login: string; password: string; fullName?: string;	response: string	Реєструє користувача в системі, надсилає у відповідь чи успішна дія
sprintById	sprintId: number;	export type Sprint = Readonly<{}> ;	Шукає спринт по ідентифікатору, повертає спринт
sprintGanttChart	sprintId: number;	user: User; notCompleted: number; total: number; tasks: Task[];	Збирає інформацію для побудови графіку спринта за алгоритмом
unassignUser	userId: number; projectId: number;	response: string	Видалити користувача з проєкту, надсилає у відповідь чи успішна дія

Продовження таблиці 4.1

Назва інтерфейсу	Вхідні параметри	Параметри відповіді	Опис інтерфейсу
usersByProject	projectId: number;	id: string; name: string; password?: string; color?: string;	Шукає користувачів, які призначені на проєкт, повертає масив користувачів

4.4 Опис звітів

Результат планування спринта для команди виконавців за допомогою Комплексу представлений такими звітами:

- календарний план виконання задач для команди - розклад виповнення задач для команди, визначений порядок виконання задач для команди в цілому;
- звіт про проведення спринта – графічне та аналітичне представлення результатів.

Вигляд календарного плану представлений на рисунку 4.2

Рисунок 4.2 – Календарний план виконання задач виконавцями.

Календарний план представлений діаграмою Гантта. Три верхніх рядки представляють часові інтервали. У першому рядку декілька колонок, кожна з яких відповідає тижням. Підпис колонки – дата з якої починається тиждень. У наступному рядку кожна колонка відповідає дню тижня. Червоним кольором підсвічені початок та кінець спринта. Зеленим підсвічений поточний день. Наступний рядок умовно поділяє робочий день навпіл (по 4 години).

Для кожного виконавця у відповідному рядку представлені задачі у послідовності, що відповідає рекомендованому порядку виконання. Кожна задача представлена блоком з кодом задачі, директивним строком, статусом. Для кожного виконавця відображається поточний прогрес роботи – кількість завершених задач відносно загальної кількості.

Вигляд звіту про проведення спринта зображений на рисунку 4.3

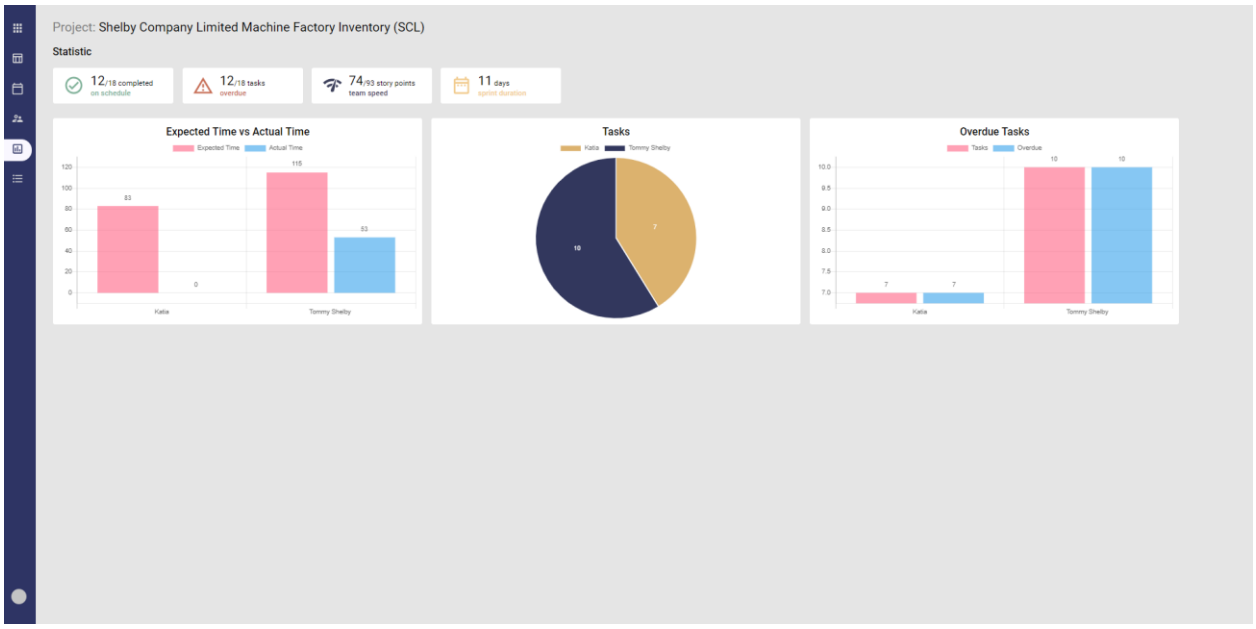


Рисунок 4.3 – Звіт про проведення спринта

Звіт за результатами спринта наявний у відповідній вкладці звітів. Він представлений віджетами та графіками.

Віджети надають таку інформацію:

- кількість завершених задач та характеристика спринта (за графіком, поза розкладом, раніше за запланований графік);

- кількість задач, виконаних із запізненням;
- швидкість команди, що вимірюється у відносній оцінці (англ. story points);
- кількість днів, які тривав спринт.

Представлені такі графіки:

- діаграма очікуваного часу проти фактичного часу (англ. Expected Time vs Actual Time) – для кожного виконавця стовпчикова діаграма (на горизонтальні осі – імена виконавців, на вертикальній осі – час у годинах);
- діаграма задач – кругова діаграма кількості задач, що реалізовані кожним виконавцем;
- діаграма запізнених задач – стовпчикова діаграма, що показує загальну кількість задач виконавця та кількість задач, які він завершив з запізненням (на горизонтальні осі – імена виконавців, на вертикальній осі – кількість у одиницях).

Висновок до розділу

У даному розділі проведено огляд технологій, що використовуються для розробки Комплексу (PostgreSQL, JavaScript, Angular, Node.js); наведено їх переваги та недоліки, та обґрунтоване використання в даній роботі. Відповідно до обраних технологій, було визначено вимоги до програмного забезпечення.

Розглянута архітектура комплексу, основні сутності, виконано детальний опис взаємодії складових архітектури. За допомогою діаграми послідовності показано процеси Комплексу; показано 2 основних сценарії роботи, а саме, авторизація та всі інші сценарії. Компоненти та їх взаємодію було представлено на діаграмі компонентів. Було наведено опис основних прикладних програмних інтерфейсів. Описано звіти та їх зміст.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

5.1.1 Ведення виконавців

Щоб зареєструватись в системі, користувачу необхідно виконати таку послідовність дій:

а) перейти за посиланням task-schedule-manager-ui.herokuapp.com;

б) система відображає початкове вікно, приклад екранних форм на рисунку 5.1;

в) ввести логін, пароль у форму та натиснути кнопку “Sign Up” (Зареєструватись);

г) система додає користувача в базу даних;

ґ) система відображає вікно проєктів користувача.

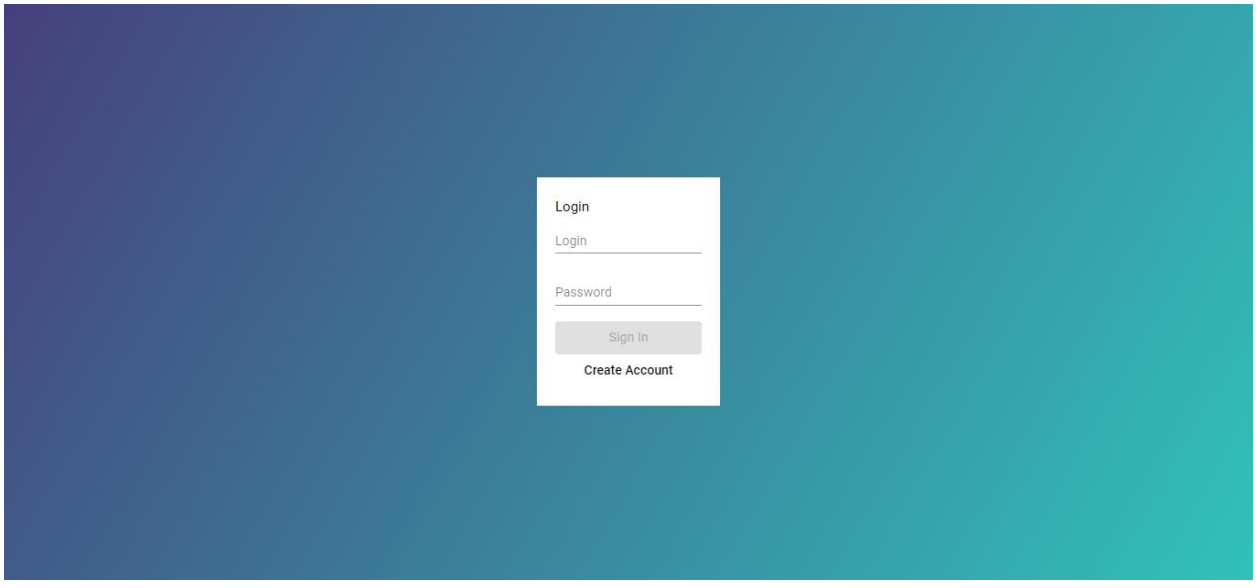


Рисунок 5.1 – Початкова сторінка системи

5.1.2 Ведення проєктів

Щоб керувати проєктами, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) система відображає картки проєктів, приклад екранних форм на рисунку 5.2;

в) щоб додати проєкт, натиснути кнопку “Add project” (додати проєкт), потім ввести необхідну інформацію у поп-апі та натиснути кнопку “Save” (зберегти);

г) щоб відредагувати проєкт, натиснути на олівець на проєкті, далі оновити інформацію у відкритому поп-апі та зберегти;

ґ) щоб видалити проєкт, натиснути на іконку відерця.

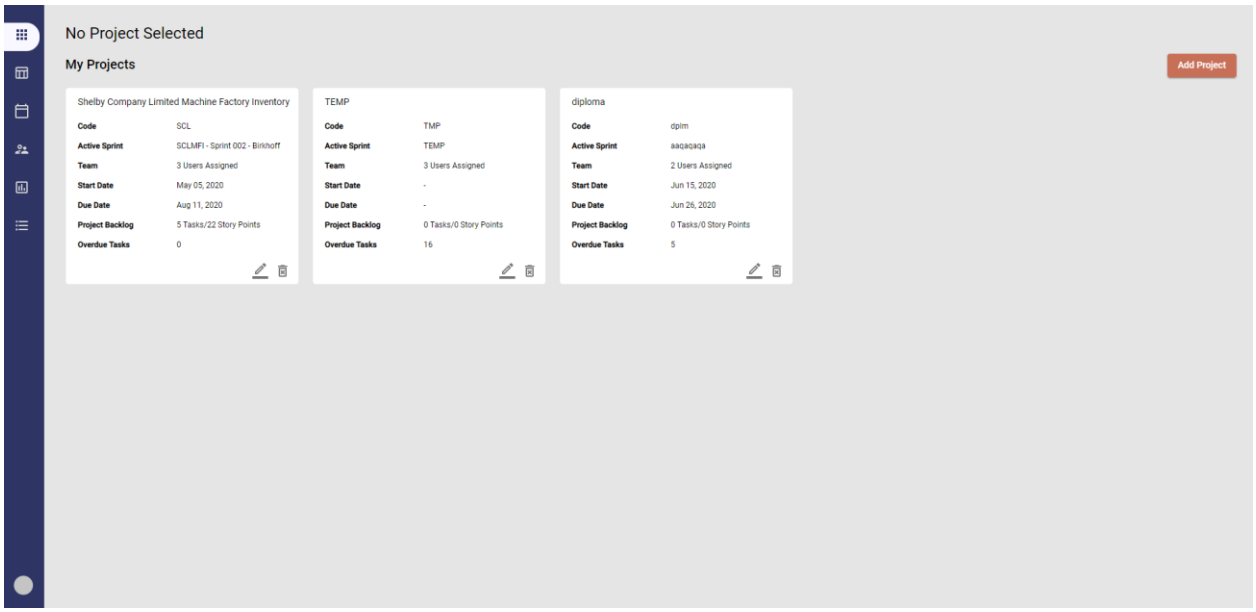


Рисунок 5.2 – Список проєктів

5.1.3 Ведення задач

Щоб створити задачу, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) на вкладці «Проєкти» обрати бажаний проєкт;

в) натиснути на іконку списку у навігаційній панелі;

г) система відображає список проєктних задач, приклад екранних форм на рисунку 5.3;

ґ) натиснути кнопку “Create task” (Створити задачу);

д) система відображає поп-ап для введення інформації про задачу (назва, директивний строк, часова оцінка, відносна оцінка, опис);

е) заповнити поля;

є) натиснути кнопку “Save” (зберегти);

ж) система закриває поп-ап;

з) система створює задачу і відображає її у списку проєктних задач.

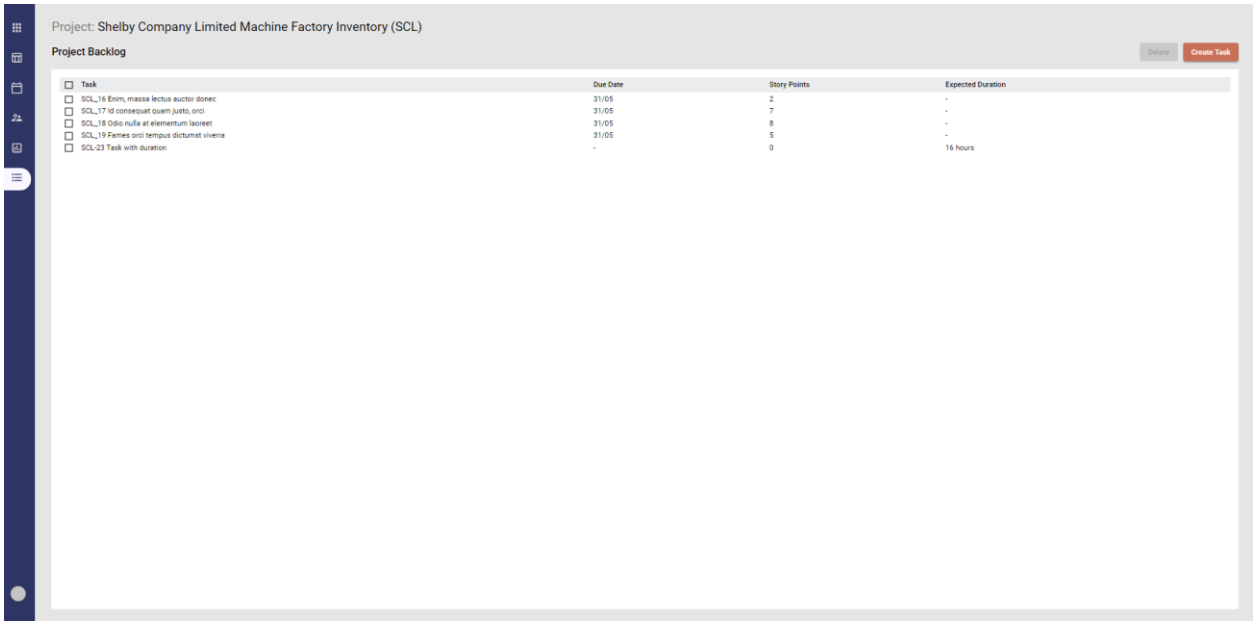


Рисунок 5.3 – Список проєктних задач

Для редагування задачі користувач має повторити дії а)-г) з алгоритму створення задачі, потім:

а) натиснути на задачу зі списку;

б) система відображає поп-ап з інформацією про задачу та редагованими полями;

в) оновити інформацію про задачу та натиснути кнопку “Save” (зберегти);

г) система зариває поп-ап;

ґ) система відображає оновлену інформацію про задачу.

Для видалення задачі користувач має повторити дії а-г з алгоритму створення задачі, потім:

а) позначити задачу за допомогою чек-боксу;

б) натиснути кнопку “Delete” (видалити);

в) система видаляє задачу зі списку.

5.1.4 Ведення проєктних команд

Щоб керувати проєктною командою, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) на вкладці «Проєкти» обрати бажаний проєкт;

в) натиснути на іконку команди у навігаційній панелі;

г) система відображає команду виконавців поточного проєкту, приклад екранних форм на рисунку 5.4;

ґ) щоб призначити виконавця на проєкт, натиснути кнопку “Manage Team” (керувати командою) та знайти потрібного користувача у відкритому поп-апі, потім натиснути кнопку “Save” (зберегти);

д) щоб видалити виконавця з проєкту, натиснути ‘x’ у відповідному рядку.

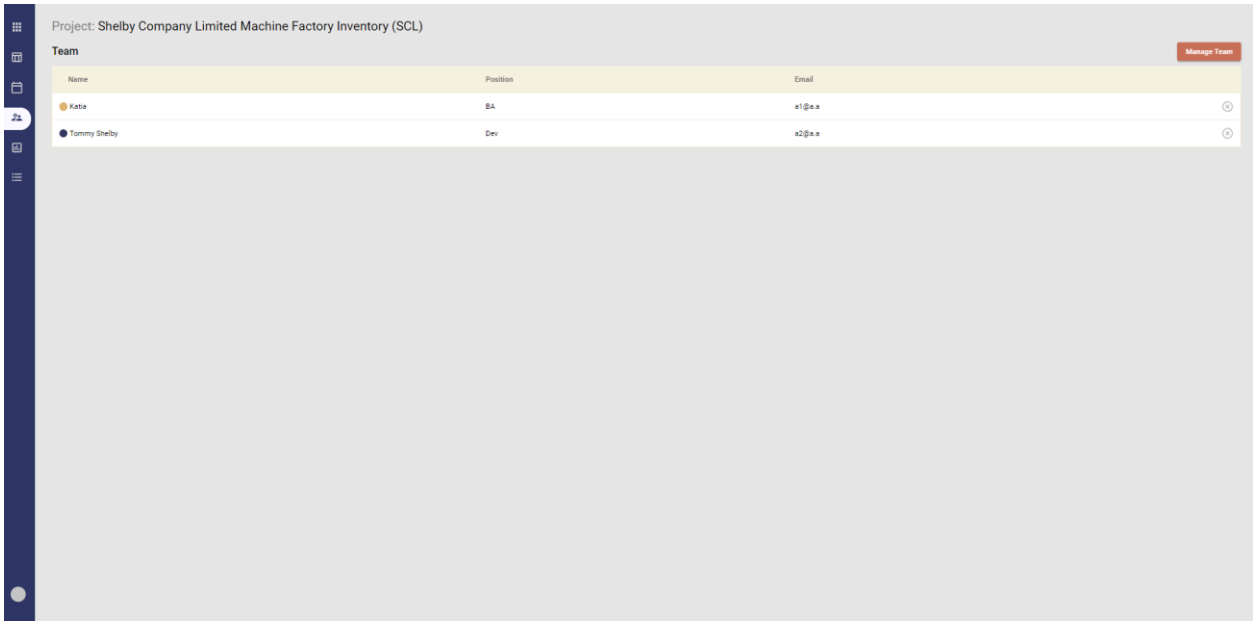


Рисунок 5.4 – Список проєктної команди виконавців

5.1.5 Ведення задач виконавців

Щоб керувати задачею, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) на вкладці «Проєкти» обрати бажаний проєкт;

в) натиснути на іконку канбан у навігаційній панелі;

г) система відображає канбан дошки з проєктними задачами, приклад екранних форм на рисунку 5.5;

ґ) натиснути на задачу з дошки “Open”, яка призначена поточному користувачу;

д) система відображає поп-ап з інформацією про задачу;

е) заповнити поля форми: “Status”: “Closed”, “Actual duration”: “1 day 4 hours”;

є) натиснути кнопку “Save” (зберегти);

ж) вікно з інформацією про задачу закрите, система перемістила задачу на дошку “Closed”.

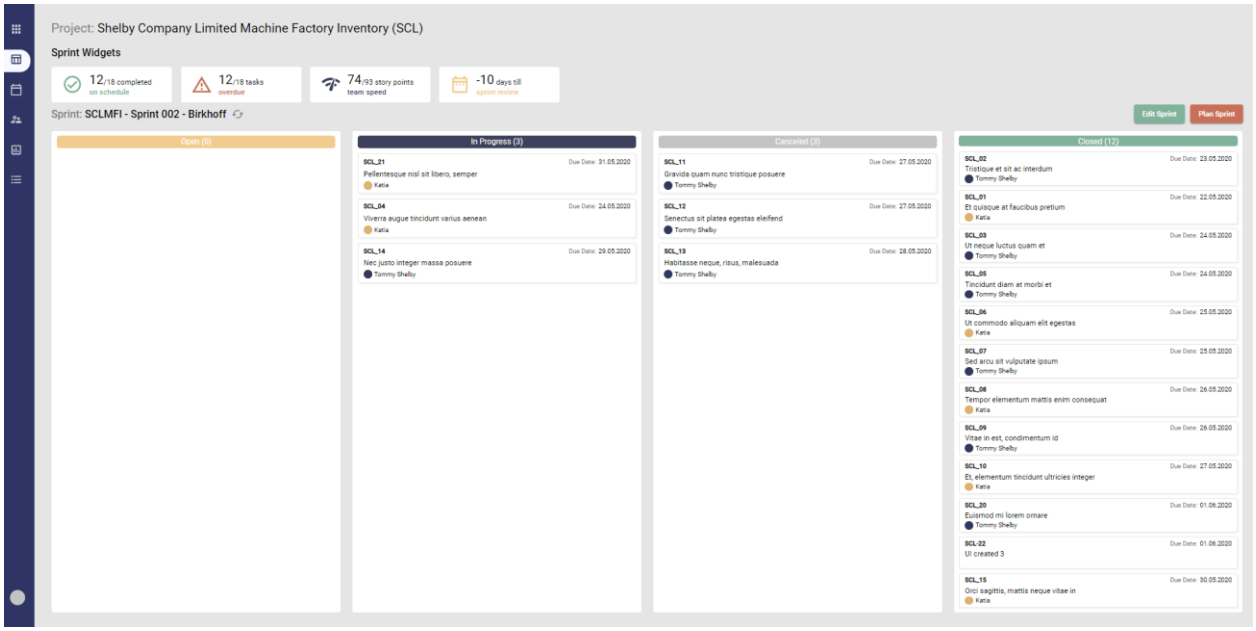


Рисунок 5.5 – Список проєктних задач

5.1.6 Складання плану робіт виконавцями

Щоб переглянути план робіт, складений для виконавців на поточний спринт, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) на вкладці «Проєкти» обрати бажаний проєкт;

в) натиснути на іконку календаря у навігаційній панелі;

г) система відображає план робіт для виконавців у вигляді діаграми Гантта, приклад екранних форм на рисунку 5.6.

Рисунок 5.6 – План робіт для виконавців

5.1.7 Формування звітності

Щоб переглянути звіти за результатами спринта, користувач повинен виконати таку послідовність дій:

а) авторизуватись в системі, ввівши свій логін та пароль;

б) на вкладі «Проєкти» обрати бажаний проєкт;

в) натиснути на іконку графіків у навігаційній панелі;

г) система відображає аналітику та графіки за результатами спринта, приклад екранних форм на рисунку 5.7.

Рисунок 5.7 – Звіт про проведення спринта

5.2 Випробування програмного продукту

В цьому підрозділі наведено опис тестів і порядок їх виконання для перевірки відповідності програмного забезпечення комплексу задач функціональним вимогам, представленим у технічному завданні на створення комплексу задач з підтримки процесу управління командою виконавців.

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач з підтримки процесу управління командою виконавців вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

В Комплексі реалізовано багато функцій, як основних так і додаткових. Оскільки розмір дипломного проєкту обмежений, проведемо тестування для основних “sunny-day” сценаріїв таких функцій:

- авторизація в системі, таблиця 5.1;
- керування проєктами (створення та видалення), таблиці 5.2-5.3;
- створення проєктних задач, таблиця 5.4;
- формування проєктної команди, таблиця 5.5;
- складання графіку виконання задач, таблиця 5.6;
- перегляд командних задач, таблиця 5.7;
- виконання задачі (переведення задачі в статус «Завершено», виставлення фактичного часу виконання), таблиця 5.8;
- перегляд звіту, таблиця 5.9;
- обробка введення некоректних даних на прикладі авторизації, таблиця 5.10.

Таблиця 5.1 – Тест функції авторизації в системі.

Назва	Тест авторизації в системі	
Функція/Use Case	Авторизація в системі	
Дія		**Очікуваний результат**
Передумова		
Відкрита початкова сторінка логіну комплексу задач		
Кроки тесту		
Заповніть логін та пароль користувача у відповідні поля форми: “login”: user1@mail.com”, “password”: “password”		Дані успішно введені
Кроки тесту		
Натисніть кнопку “Log in”		Користувача авторизовано в системі. Відкрита сторінка проєктів користувача

Продовження таблиці 5.1

Дія	**Очікуваний результат**
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.2 – Тест функції створення проєктів.

Назва	Тест створення проєктів
Функція/Use Case	Створення проєктів
Дія	**Очікуваний результат**
Передумова	
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача	
Кроки тесту	
Натисніть кнопку “Add project”	Відкрите поп-ап вікно створення проєкту з можливістю введення даних
Заповніть форму створення проєкту: “Name”: “Billing Project” “Code”: “BP” “Start Date”: “01.06.2020” “Due Date”: “01.09.2020”	Дані успішно введені
Натисніть кнопку “Save”	Вікно створення проєкту закрите. Проєкт додано до списку проєктів користувача. Проєкт відображається на екрані.
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**
Назва	Тест видалення проєктів
Функція/Use Case	Видалення проєктів

Таблиця 5.3 – Тест функції видалення проєктів.

Дія	**Очікуваний результат**
Передумова	
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача.	
Кроки тесту	
Натисніть кнопку “відерце” (у вигляді іконки) біля проєкту “Billing Project”	Проєкт видалено зі списку проєктів користувача. Проєкт не відображається на екрані.
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.4 – Тест функції створення проєктних задач.

Назва	Тест створення проєктних задач
Функція/Use Case	Формування резерву проєктних задач
Дія	**Очікуваний результат**
Передумова	
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача.	
Кроки тесту	
Натисніть кнопку «Задачі» у навігаційній панелі	Відкрита сторінка задач проєкту “Billing Project”.
Натисніть кнопку “Create task”	Відкрите поп-ап вікно створення задачі з можливістю введення даних
Заповніть форму створення задачі: “Name”: “Create Billing component” “Due Date”: “05.06.2020” “Expected Duration”: “2 days 0 hours” “Story points”: “16” “Description”: “Please, implement billing component”	Дані успішно введені

Продовження таблиці 5.4

Дія	**Очікуваний результат**
Натисніть кнопку “Save”	Вікно створення задачі закрите. Задачу додано до списку проєктних задач. Задача відображається на екрані у списку задач.
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.5 – Тест функції формування проєктної команди.

Назва	Тест формування проєктної команди	
Функція/Use Case	Формування проєктної команди	
Дія		**Очікуваний результат**
Передумова		
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача.		
Кроки тесту		
Натисніть кнопку «Команда» у навігаційній панелі		Відкрита сторінка команди проєкту “Billing Project”.
Натисніть кнопку “Manage Team”		Відкрите поп-ап вікно додавання користувача до команди з можливістю введення даних
Заповніть форму створення задачі: “Users”: “John Snow” обрати з випадаючого списку		Дані успішно введені
Натисніть кнопку “Save”		Вікно додавання користувача закрите. Користувача додано до списку проєктної команди. Користувач відображається на екрані у списку команди.
Післяумова		
Натисніть на кнопку “Log out”		Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)		**Пройдений**

Таблиця 5.6 – Тест функції складання графіку виконання задач.

Назва	Тест складання графіку виконання задач
Функція/Use Case	Формування резерву спринта; Перегляд графіку виконання задач
Дія	**Очікуваний результат**
Передумова	
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача. Користувача додано до проєктної команди “Billing Project”. Проєктні задачі існують.	
Кроки тесту	
Натисніть кнопку «Канбан» у навігаційній панелі	Відкрита сторінка канбан дошок проєкту “Billing Project”.
Натисніть кнопку «Configure Sprint»	Відкрите поп-ап вікно налаштування спринта з можливістю введення даних.
Заповніть форму налаштування спринта: “Name”: “BP – Sprint 001 - Apollo” “Start Date”: “01/06/2020” “Due Date”: “14/06/2020” Обрати всі задачі зі списку задач в таблиці	Дані успішно введені
Натисніть кнопку “Save”	Вікно налаштування спринта закрите. Обрані задачі з’явились на канбан дошках. Система відображає назву спринта.
Натисніть кнопку “Plan Sprint”	Відкрита сторінка плану спринта. Система відображає план у вигляді діаграми Гантта.
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.7 – Тест функції перегляду командних задач.

Назва	Тест перегляду командних задач	
Функція/Use Case	Перегляд командних задач	
Дія		**Очікуваний результат**
Передумова		
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача. Користувача додано до проєктної команди “Billing Project”. Проєктні задачі існують та додані у поточний спринт. Поточний спринт запланований.		
Кроки тесту		
Натисніть кнопку «Канбан» у навігаційній панелі		Відкрита сторінка канбан дошок проєкту “Billing Project”. Система відображає 4 канбан дошки “Open”, “In Progress”, “Canceled”, “Closed”, в яких містяться командні задачі з відповідними актуальними статусами. На кожній задачі відображається ім’я виконавця.
Післяумова		
Натисніть на кнопку “Log out”		Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)		**Пройдений**

Таблиця 5.8 – Тест функції виконання задачі.

Назва	Тест виконання задачі	
Функція/Use Case	Фіксування поточного стану задачі; Фіксування витраченого часу на виконання задачі	
Дія		**Очікуваний результат**
Передумова		
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача. Користувача додано до проєктної команди “Billing Project”. Проєктні задачі існують та додані у поточний спринт. Поточний спринт запланований. Задача “Create Billing component” призначена поточному користувачу.		

Продовження таблиці 5.8

Дія	**Очікуваний результат**
Кроки тесту	
Натисніть кнопку «Канбан» у навігаційній панелі	Відкрита сторінка канбан дошок проєкту “Billing Project”. Задача “Create Billing component” відображається на дошці “In Progress”.
Натисніть на задачу “Create Billing component”.	Відкрите поп-ап вікно з інформацією про задачу.
Заповніть поля форму: “Status”: “Closed” “Actual duration”: “1 day 4 hours”	Дані успішно введені
Натисніть кнопку “Save”	Вікно з інформацією про задачу закрите. Система перемістила задачу на дошку “Closed”.
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.9 – Тест функції перегляду звіту.

Назва	Тест перегляду звіту
Функція/Use Case	Перегляд проєктних звітів
Дія	**Очікуваний результат**
Передумова	
Користувача авторизовано в системі. Відкрита сторінка проєктів користувача. Проєкт “Billing Project” створено та він відображається у списку проєктів користувача. Користувача додано до проєктної команди “Billing Project”. Проєктні задачі існують та додані у поточний спринт. Поточний спринт заплановано.	
Кроки тесту	
Натисніть кнопку «Графік» у навігаційній панелі	Відкрита сторінка звітів. Система відображає кількість завершених задач за спринт, кількість задач, виконаних з запізненням, сумарну відносну оцінку, кількість днів спринта; діаграму згорання задач, діаграму очікуваного часу проти фактичного часу, діаграму задач, діаграму запізнених задач.

Продовження таблиці 5.9

Дія	**Очікуваний результат**
Післяумова	
Натисніть на кнопку “Log out”	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Таблиця 5.10 – Тест функції обробка введення некоректних даних на прикладі авторизації.

Назва	Тест обробка введення некоректних даних на прикладі авторизації
Функція/Use Case	Авторизація в системі
Дія	**Очікуваний результат**
Передумова	
Відкрита початкова сторінка логіну комплексу задач	
Кроки тесту	
Заповніть логін та пароль користувача у відповідні поля форми: “login”: “user1@mail.com” “password”: “I don’t know”	Дані успішно введені
Натисніть кнопку “Log in”	Користувача не авторизовано в системі. Виведено повідомлення “User is not found. Please, check you credentials”.
Післяумова	
Оновіть сторінку	Відкрита початкова сторінка логіну
Результат тесту (пройдений, провалений, заблокований)	**Пройдений**

Висновок до розділу

У даному розділі надано керівництво користувача для взаємодії з комплексом. Описано послідовність дій для таких сценаріїв:

- реєстрація в системі;
- керування проєктами;

- створення, редагування та видалення задач;
- призначення та видалення користувачів з проєктів;
- перегляд плану реалізації задач;
- перегляд звітності.

Програмний продукт було протестовано за наведеними вище сценаріями. Комплекс показав, що очікуваний результат співпадає з дійсним результатом роботи.

ЗАГАЛЬНІ ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано та розроблено комплекс задач з підтримки процесу управління командою виконавців. Розкрита предметна область команд, які використовують методологію розробки Scrum, розглянуто ролі користувачів та процеси їхньої взаємодії.

Виявлені аналоги комплексу задач, проведено їх порівняння та визначено основну перевагу комплексу – складання графіку виконання задач.

Було запропоновано наближений алгоритм для складання розкладу виконання робіт з різними директивними строками ідентичними машинами за двома критеріями. Алгоритм було розроблено на основі результатів близьких задач. Було досліджено умови, за яких можна отримати допустимий розклад без запізнень, для якого було запропоновано показник жорсткості директивних строків.

Розглянута архітектура комплексу, основні сутності, виконано детальний опис взаємодії складових архітектури. За допомогою діаграми послідовності показано процеси Комплексу; показано 2 основних сценарії роботи, а саме, авторизація та всі інші сценарії. Компоненти та їх взаємодію було представлено на діаграмі компонентів. Було наведено опис основних прикладних програмних інтерфейсів. Описано звіти та їх зміст.

Надано керівництво користувача для взаємодії з комплексом. Програмний продукт було протестовано, комплекс показав, що очікуваний результат співпадає з дійсним результатом роботи.

Подальші кроки для розвитку ідеї комплексу:

- розділення ролей в системі з розробкою матриці безпеки;
- розширення функції керування задачею;
- реалізація бізнес календаря для планування Scrum-зустрічей;
- додавання модуля зі Scrum принципами;

- додавання можливості інтеграції задач із зовнішніх систем;
- додавання можливості плагінізації модуля планування графіку задач у зовнішні системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Авраменко К.А., Жданова О.Г. Про двокритеріальну задачу складання розкладу виконання робіт з різними директивними строками ідентичними машинами // Матеріали IV всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2020) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», – 2020. 24, 30 квітня 2020р. – 182 с.

2. Сазерленд Д. Scrum. Навчись робити вдвічі більше за менший час / Джефф Сазерленд., 2016. – 280 с.

3. Atlassian knowledge base [Електронний ресурс] // Confluence. – 2020. – Режим доступу до ресурсу: https://confluence.atlassian.com/kb.

4. Trello help [Електронний ресурс] – Режим доступу до ресурсу: https://help.trello.com.

5. Hey Space Knowledge Base [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://heyspace.user.com/knowledge-base/root/.

6. Tuleap documentation [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.tuleap.org.

7. Pinedo Michael L Planing and Scheduling, pringer Science Business Media, 2009 - 532 с.

8. Конвей Р.В., Максвелл В.Л., Миллер Л.В. Теория расписний. – М.: Наука, 1975.

9. J.R. Jackson (1955) “Scheduling a Production Line to Minimize Maximum Tardiness”, Research Report 43, Management Science Research Project, University of California, Los Angeles.

10. W.E. Smith (1956) “Various Optimizers for Single Stage Production”, Naval Research Logistics Quarterly, Vol. 3.

11. Сперкач М.О. Задача визначення максимально пізнього моменту початку виконання завдань із спільним жорстким директивним терміном паралельними пристроями різної продуктивності // Вісник НТУУ “КПІ”.

Серія «Інформатика, управління та обчислювальна техніка». – К.: “ВЕК+”, 2015. – №63 – С.12-18

12. Node.js Starter Kit [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://github.com/HowProgrammingWorks/NodejsStarterKit.

13. Guides [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://material.angular.io/guides.

14. A Reactive State Management Tailored-Made for JS Applications [Електронний ресурс] – Режим доступу до ресурсу: https://datorama.github.io/akita/.

15. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.

16. McMillan M. Data Structures and Algorithms with JavaScript / Michael McMillan., 2014. – 252 с.

17. Фленаган Д. JavaScript: The Definitive Guide / Девід Фленаган., 2012. – 1080 с. – (Бестселеры O'Reilly).

18. Introduction to the Angular Docs [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://angular.io/docs.

19. Docs Node.js [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://nodejs.org/en/docs/.

ДОДАТОК А

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1003882170

Дата перевірки:
08.06.2020 18:29:25 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2020 17:45:01 EEST

ID користувача:
77149

Назва документу: **Avramenko_bachelor_is63**

ID файлу: **1003897026** Кількість сторінок: **75** Кількість слів: **10937** Кількість символів: **89534** Розмір файлу: **2.70 MB**

7.31% Схожість

Найбільша схожість: **2.19%** з джерело бібліотеки. ID файлу: **1000017966**

3.51% Схожість з Інтернет джерелами	34	Page 77
6.97% Текстові збіги по Бібліотеці акаунту	250	Page 77

3.49% Цитат

Цитати	25	Page 78

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	5

ДОДАТОК Б

```
async ({ sprintId }) => {
  const projectId = (await application.db.select('sprints', ['project_id'], { id: sprintId
}))[0].projectId;
  const whereUserId = `WHERE id IN
                      (
                        SELECT user_id
                        FROM users_projects
                        WHERE project_id = ${projectId}
                      )`;
  const users = await application.db.select(`system_users ${whereUserId}`, ['id']);
  let tasks = await application.db.select('tasks', ['id', 'estimate', 'due_date'], { sprintId });
  const userTaskMap = new Map();
  users.forEach(user => userTaskMap.set(user.id, []));
  tasks.sort((a, b) => b.dueDate - a.dueDate);
  const summaryDuration = tasks.reduce((result, next) => result + next.estimate, 0);
  const perfectDuration = Math.ceil(summaryDuration / users.length);
  const userCapacity = new Map();
  users.forEach(user => userCapacity.set(user.id, perfectDuration));
  const taskLength = tasks.length;
  for (let i = 0; i < taskLength; i++) {
    let maxCapacity = 0;
    let maxCapacityUser = users[0].id;
    Array.from(userCapacity.entries()).forEach(([userId, capacity]) => {
      if (capacity > maxCapacity) {
        maxCapacity = capacity;
        maxCapacityUser = userId;
      }
    });
    const properTasks = tasks.filter(task => task.dueDate > maxCapacity);
    const maxEstimateTask = properTasks.reduce((prev, next) => (prev.estimate >= next.estimate
? prev : next));
    userTaskMap.get(maxCapacityUser).push(maxEstimateTask.id);
    tasks = tasks.filter(task => task.id !== maxEstimateTask.id);
    userCapacity.set(maxCapacityUser, maxCapacity - maxEstimateTask.estimate);
  }
  console.log(userTaskMap);
  const userPromises = Array.from(userTaskMap.entries())
    .map(([userId, taskIds]) => application.db.update('tasks', { userId }, { id: taskIds }));
  await Promise.all(userPromises);
  const taskPromises = [];
  Array.from(userTaskMap.values()).forEach(tasks => {
    tasks.reverse().forEach((task, index) => {
      taskPromises.push(application.db.update('tasks', { orderNumber: index + 1 }, { id: task }));
    });
```

```
  });
  await Promise.all(taskPromises);
  await application.db.update('sprints', { readonly: true }, { id: sprintId });
  return { result: 'success' };
};


'use strict';
const { Pool } = require('pg');
const toSnakeCase = require('../utils/toSnakeCase.js');
const fromSnakeCase = require('../utils/fromSnakeCase');
const OPERATORS = ['>=', '<=', '<>', '>', '<'];
const where = (conditions, firstArgIndex = 1) => {
  conditions = toSnakeCase(conditions);
  const clause = [];
  const args = [];
  let i = firstArgIndex;
  const keys = Object.keys(conditions);
  for (const key of keys) {
    let argNum = ` $${i}`;
    let operator = '=';
    let value = conditions[key];
    if (typeof value === 'string') {
      for (const op of OPERATORS) {
        const len = op.length;
        if (value.startsWith(op)) {
          operator = op;
          value = value.substring(len);
        }
      }
      if (value.includes('*') || value.includes('?')) {
        operator = 'LIKE';
        value = value.replace(/\*/g, '%').replace(/\?/g, '_');
      }
    } else if (Array.isArray(value)) {
      operator = `= ANY($${i}::int[])`;
      argNum = '';
    }
    clause.push(`${key} ${operator}${argNum}`);
    args.push(value);
    i++;
  }
  return { clause: clause.join(' AND '), args };
};
const updates = (delta, firstArgIndex = 1) => {
  delta = toSnakeCase(delta);
  const clause = [];
```

```
  const args = [];
  let i = firstArgIndex;
  const keys = Object.keys(delta);
  for (const key of keys) {
    const value = delta[key].toString();
    clause.push(`${key} = $${i++}`);
    args.push(value);
  }
  return { clause: clause.join(', '), args };
};

class Database {
  constructor(config, application) {
    this.pool = new Pool(config);
    this.application = application;
  }
  query(sql, values) {
    const data = values ? values.join(',') : '';
    this.application.logger.log(`${sql}\t[${data}]`);
    return this.pool.query(sql, values);
  }

  insert(table, record) {
    record = toSnakeCase(record);
    const keys = Object.keys(record);
    const nums = new Array(keys.length);
    const data = new Array(keys.length);
    let i = 0;
    for (const key of keys) {
      data[i] = record[key];
      nums[i] = `$${++i}`;
    }
    const fields = keys.join(', ');
    const params = nums.join(', ');
    const sql = `INSERT INTO ${table} (${fields}) VALUES (${params}) RETURNING *`;
    return this.query(sql, data);
  }
  async select(table, fields = ['*'], conditions = null) {
    conditions = toSnakeCase(conditions);
    const keys = fields.join(', ');
    const sql = `SELECT ${keys} FROM ${table}`;
    let whereClause = '';
    let args = [];

   if (conditions) {
      const whereData = where(conditions);
      whereClause = ' WHERE ' + whereData.clause;
```

```
      args = whereData.args;
    }
    const res = await this.query(sql + whereClause, args);
    return res.rows.map(row => fromSnakeCase(row));
  }

  delete(table, conditions = null) {
    conditions = toSnakeCase(conditions);
    const { clause, args } = where(conditions);
    const sql = `DELETE FROM ${table} WHERE ${clause}`;
    return this.query(sql, args);
  }

  update(table, delta = null, conditions = null) {
    conditions = toSnakeCase(conditions);
    const upd = updates(delta);
    const cond = where(conditions, upd.args.length + 1);
    const sql = `UPDATE ${table} SET ${upd.clause} WHERE ${cond.clause}`;
    const args = [...upd.args, ...cond.args];
    return this.query(sql, args);
  }

  close() {
    this.pool.end();
  }
}
module.exports = Database;
async ({ sprintId }) => {
                                    const sql =
                                      `SELECT u.id        as user_id,
                                            u.full_name as user_full_name,
                                            u.color     as color,
                                            (
                                              SELECT COUNT(id)
                                              FROM tasks
                                              WHERE sprint_id = $1
                                                AND user_id = u.id
                                                AND status IN ('closed', 'canceled')
                                            )           as completed,
                                            (
                                              SELECT COUNT(id)
                                              FROM tasks
                                              WHERE sprint_id = $1
                                                AND user_id = u.id
                                            )           as total,
                                            t.*
                                          FROM system_users u
```

```
                JOIN tasks t ON t.user_id = u.id
            WHERE t.sprint_id = $1;`;
  const queryResult = await application.db.query(sql, [sprintId]);
  const rows = (queryResult && queryResult.rows ||
[]).map(api.fromSnakeCase);
  const byUserId = new Map();
  rows.forEach(row => byUserId.set(row.userId,
(byUserId.get(row.userId) || []).concat([row])));
  const data = Array.from(byUserId.entries())
    .map(([userId, rows]) => {
      return {
        user: {
          id: userId,
          fullName: rows[0] && rows[0].userFullName,
          color: rows[0] && rows[0].color
        },
        completed: rows[0] && rows[0].completed || 0,
        total: rows[0] && rows[0].total || 0,
        tasks: rows
          .map(row => {
            return {
              ...row,
              userId: undefined,
              userFullName: undefined,
              total: undefined,
              notCompleted: undefined
            };
          })
          .sort((a, b) => a.orderNumber - b.orderNumber)
      };
    });
  return data;
};

async ({ sprintId, taskIds }) => {
                  await application.db.update('tasks', { sprintId }, { id:
                taskIds });
                  return { result: 'success' };
                };

async project => {
              delete project.id;
              const result = await application.db.insert('projects', project);
              const projectId = result.rows[0].id;
              await application.db.insert('users_projects', {
                userId: project.ownerId,
                projectId
```

```
  });
  console.log(result);
  return { result: 'success', id: projectId };
};

async ({ sprintId }) => {
  const projectId = (await application.db.select('sprints', ['project_id'],
{ id: sprintId }))[0].projectId;
  const whereUserId = `WHERE id IN
    (
     SELECT user_id
     FROM users_projects
     WHERE project_id = ${projectId}
    )`;
  const users = await application.db.select(`system_users
${whereUserId}`, ['id']);
  let tasks = await application.db.select('tasks', ['id', 'estimate',
'due_date'], { sprintId });
  const userTaskMap = new Map();
  users.forEach(user => userTaskMap.set(user.id, []));
  tasks.sort((a, b) => b.dueDate - a.dueDate);
  const summaryDuration = tasks.reduce((result, next) => result +
next.estimate, 0);
  const perfectDuration = Math.ceil(summaryDuration /
users.length);
  const userCapacity = new Map();
  users.forEach(user => userCapacity.set(user.id, perfectDuration));


  const taskLength = tasks.length;
  for (let i = 0; i < taskLength; i++) {
   let maxCapacity = 0;
   let maxCapacityUser = users[0].id;
   Array.from(userCapacity.entries()).forEach(([userId, capacity]) =>
{
    if (capacity > maxCapacity) {
     maxCapacity = capacity;
     maxCapacityUser = userId;
    }
   });
   const properTasks = tasks.filter(task => task.dueDate >
maxCapacity);
   const maxEstimateTask = properTasks.reduce((prev, next) =>
(prev.estimate >= next.estimate ? prev : next));
   userTaskMap.get(maxCapacityUser).push(maxEstimateTask.id);
   tasks = tasks.filter(task => task.id !== maxEstimateTask.id);
   userCapacity.set(maxCapacityUser, maxCapacity -
```

```
                maxEstimateTask.estimate);
                }
                console.log(userTaskMap);


                const userPromises = Array.from(userTaskMap.entries())
                  .map(([userId, taskIds]) => application.db.update('tasks', { userId
              }, { id: taskIds }));
                await Promise.all(userPromises);
                const taskPromises = [];
                Array.from(userTaskMap.values()).forEach(tasks => {
                  tasks.reverse().forEach((task, index) => {
                    taskPromises.push(application.db.update('tasks', { orderNumber:
              index + 1 }, { id: task }));
                  });
                });
                await Promise.all(taskPromises);
                await application.db.update('sprints', { readonly: true }, { id:
              sprintId });


                return { result: 'success' };
              };

async task => {
                delete task.id;
                const projectId = task.projectId;
                if (projectId) {
                  const project = await application.db.select('projects', ['code'], { id: projectId
              });
                  const taskCount = await application.db.select('tasks', ['COUNT(id) as
              count'], { projectId });
                  task.code = `${project[0].code}-${++taskCount[0].count}`;
                }
                await application.db.insert('tasks', task);
                return { result: 'success' };
              };

async ({ projectId, sprintId }) => {
                  const sprintIdWhere = sprintId ? `sprint_id IS NULL OR
                sprint_id = ${sprintId}` : 'sprint_id IS NULL';
                  const where = `WHERE project_id = ${projectId} AND
                status = 'open' AND (${sprintIdWhere}) ORDER BY
                code`;
                  const data = await application.db.select(`tasks ${where}`,
                undefined);
                  return data;
```

```
                                        };

async ({ userId }) => {
                                  const sql =
                                     `SELECT p.id as id, p.name as name, p.code as code, s.name as
                                sprint, p.active_sprint_id,
                                        p.start_date as start_date, p.due_date as due_date, s.code as
                                sprint_code,
                                     (
                                          SELECT COUNT(up.user_id)
                                          FROM users_projects up
                                          WHERE up.project_id = p.id)
                                     as user_count,
                                     (
                                          SELECT COUNT(t.id)
                                          FROM tasks t
                                          WHERE t.project_id = p.id
                                               AND t.sprint_id is NULL
                                               AND t.status = 'open')
                                     as backlog_count,
                                     (
                                          SELECT SUM(t.story_points)
                                          FROM tasks t
                                          WHERE t.project_id = p.id
                                               AND t.sprint_id is NULL
                                               AND t.status = 'open')
                                     as story_points,
                                     (
                                          SELECT COUNT(t.id)
                                          FROM tasks t
                                          WHERE t.project_id = p.id
                                               AND t.status IN ('open', 'in_progress')
                                               AND t.due_date::date > CURRENT_DATE)
                                     as overdues
                                    FROM projects p LEFT JOIN sprints s on (p.active_sprint_id =
                                s.id) JOIN users_projects up ON up.project_id=p.id
                                     WHERE up.user_id = $1;`;
                                  const data = await application.db.query(sql, [userId]);
                                  return data && (data.rows || []).map(api.fromSnakeCase);
                                };

                          async ({ login, password, fullName, color, email, position }) => {
                          const hash = await api.security.hashPassword(password);
                            await application.auth.registerUser(login, hash, fullName);
                            await application.db.update('system_users', {
                              color,
                              email,
```

```
            position
          }, { login });
          return { result: 'success' };
        };
async () => {
  const status = context.token ? 'logged' : 'not logged';
  return { result: status };
};
async ({ sprintId }) => {
  const data = await application.db.select('sprints', undefined, { id: sprintId });
  return data[0];
};
async () => {
  return { result: 'success' };
};
async project => {
  const where = { id: project.id };
  delete project.id;
  await application.db.update('projects', project, where);
  return { result: 'success' };
};
async sprint => {
  const where = { id: sprint.id };
  const sprintId = sprint.id;
  delete sprint.id;
  const newTasks = sprint.tasks || [];
  delete sprint.tasks;
  await application.db.update('sprints', sprint, where);
  const oldTasks = (await application.db.select('tasks', ['id'], {
    sprintId,
    status: 'open'
  })).map(item => item.id);
  console.log(oldTasks);
  console.log(newTasks);
  const tasksToDelete = oldTasks.filter(taskId => !newTasks.includes(taskId));
  const taskToUpdate = newTasks.filter(taskId => !oldTasks.includes(taskId));
  console.log(tasksToDelete);
  console.log(taskToUpdate);
  if (tasksToDelete.length) {
    const inClause = tasksToDelete.reduce((result, next, index) => {
      if (index > 0) {
        result = `${result}, `;
      }
      result = `${result}${next}`;
      return result;
    }, '');
    const deleteSql = `UPDATE tasks SET sprint_id = NULL where id IN (${inClause})`;
```

```
    await application.db.query(deleteSql);
  }
  if (taskToUpdate.length) {
    await application.db.update('tasks', { sprintId }, { id: taskToUpdate });
  }
  return { result: 'success' };
};
async task => {
  const where = { id: task.id };
  delete task.id;
  await application.db.update('tasks', task, where);
  return { result: 'success' };
};
async ({ sprintId }) => {
  const sql = `UPDATE projects SET active_sprint_id = NULL WHERE active_sprint_id =
${sprintId}`;
  await application.db.query(sql);
  return { result: 'success' };
};
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Олена ЖДАНОВА*
(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“14” квітня 2020 р.

Комплекс задач з підтримки процесу управління командою виконавців

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6101.01.000 ТЗ*

на 8 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Комплекс задач з підтримки процесу управління командою виконавців, далі – Комплекс.

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником системи є ТОВ «НЕТКРЕКЕР». Розробником системи є Авраменко Катерина Анатоліївна.

1.3 Перелік документів, на підставі яких створюється система

Підставою для розробки Комплексу є завдання на переддипломну практику.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку робіт по створенню Комплексу – 17.05.2020. Плановий робіт завершення робіт по створенню Комплексу – 15.06.2020.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ КОМПЛЕКСУ ЗАДАЧ

2.1 Призначення системи

Комплекс задач призначений для підтримки процесу управління командою виконавців, а саме:

- для розподілу задач між учасниками виробництва;
- для відстеження поточного стану задач;
- для визначення загальної кількості зроблених задач виконавцями за період часу, кількості задач, закінчених невчасно;
- для управління задачею виконавцями: зміна статусу задачі, у відповідності з її поточним станом.

2.2 Цілі створення системи

Метою даного комплексу задач є підвищення ефективності роботи команди виконавців, за рахунок максимального дотримання директивних строків та мінімізації середнього часу виконання робіт.

Для досягнення поставленої мети необхідно реалізувати такі задачі:

- ведення виконавців;
- ведення проєктів;
- ведення задач;
- ведення проєктних команд;
- ведення задач виконавців;
- складання плану робіт виконавцями;
- формування звітності.

Комплекс задач може застосовуватись як окрема система планування роботи команди так і плагін до відомих систем командної розробки (наприклад, Atlassian Jira).

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об’єктом автоматизації є процес управління проєктом командою виконавців. Для управління процесами розробки в команді використовується Scrum. Основні ролі в системі – власник проєкту, Scrum-майстер, команда виконавців.

Власник проєкту володіє баченням концепту кінцевого результату розробки, тому він приймає безпосередню участь у формуванні та пріоритезації беклогу проєкту – перелік усіх задач, які необхідно реалізувати, щоб розробити проєкт.

Scrum-майстер неформальний лідер команди виконавців. Помилково казати, що Scrum-майстер розподіляє задачі між виконавцями, контролює результати та строки. Його основний обов’язок – навчати команду принципам Scrum та впроваджувати Scrum-практики.

Команда – це люди, які будуть займатися виконанням задач. Scrum передбачає, що команди автономні. Фактично, у Scum команд відсутні керівники, адже ці команди самоорганізовані та здатні приймати рішення про свою роботу самостійно.

Команди виконавців Scrum самокеровані, до їх обов’язків можна віднести:

- розподіл та делегування поточних задач між виконавцями;
- виконання задач та надання інформації про актуальний стан задачі;
- відстеження стану проєкту;
- визначення загальної кількості зроблених задач виконавцями за період часу, кількості задач, закінчених невчасно.

4 ВИМОГИ ДО ПРОГРАМИ

4.1 Вимоги до функціональних характеристик

Функціональні характеристики комплексу:

- функція ведення виконавців;
- функція ведення проєктів;
- функція ведення задач;
- функція ведення проєктних команд;
- функція ведення задач виконавців;
- функція складання графіку виконання задач для кожного виконавця;
- функція визначення загальної кількості виконаних задач за період часу, кількості задач, закінчених невчасно;
- функція фіксації виконавцями поточного стану задачі та скільки часу було витрачено на задачу.

4.2 Вимоги до надійності

Вимоги до відмовостійкості:

- комплекс має оброблювати всі виключні ситуації, пов’язані з некоректно отриманими даними та виводити повідомлення про тип помилки.

4.3 Вимоги до складу і параметрів технічних засобів

Комплекс рекомендовано використовувати на комп’ютері, що має такі характеристики:

- операційна система – Windows 7/8/10;
- процесор – не нижче Intel Pentium CPU N3700 1.6 ГГц;
- оперативна пам’ять – 1 Гб;
- вільне місце жорсткого диску – 5 Гб.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

1. Огляд рекомендованих джерел літератури.
2. Перегляд існуючих аналогів.
3. Постановка задачі.
4. Розробка математичної частини.
5. Розробка тестових сценаріїв.
6. Розробка програмної частини.
7. Налагодження та тестування програмної частини.
8. Оформлення документації.
9. Демонстрація працюючого програмного продукту.

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Прийом програмного продукту має проводитись не пізніше, ніж 01.06.2020. Перелік необхідних матеріалів, що мають бути надані під час прийому:

- технічне завдання;
- програмний продукт та сценарії використання;
- керівництво користувача.

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603-92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

6.1 Види випробувань

Етапи прийому:

1) ознайомчий;
    a. перевірка комплектності програмної і технічної складових наданих матеріалів;
2) випробування.
    a. перевірка відповідності системи описаним вимогам.

Функції, що підлягають перевірці:

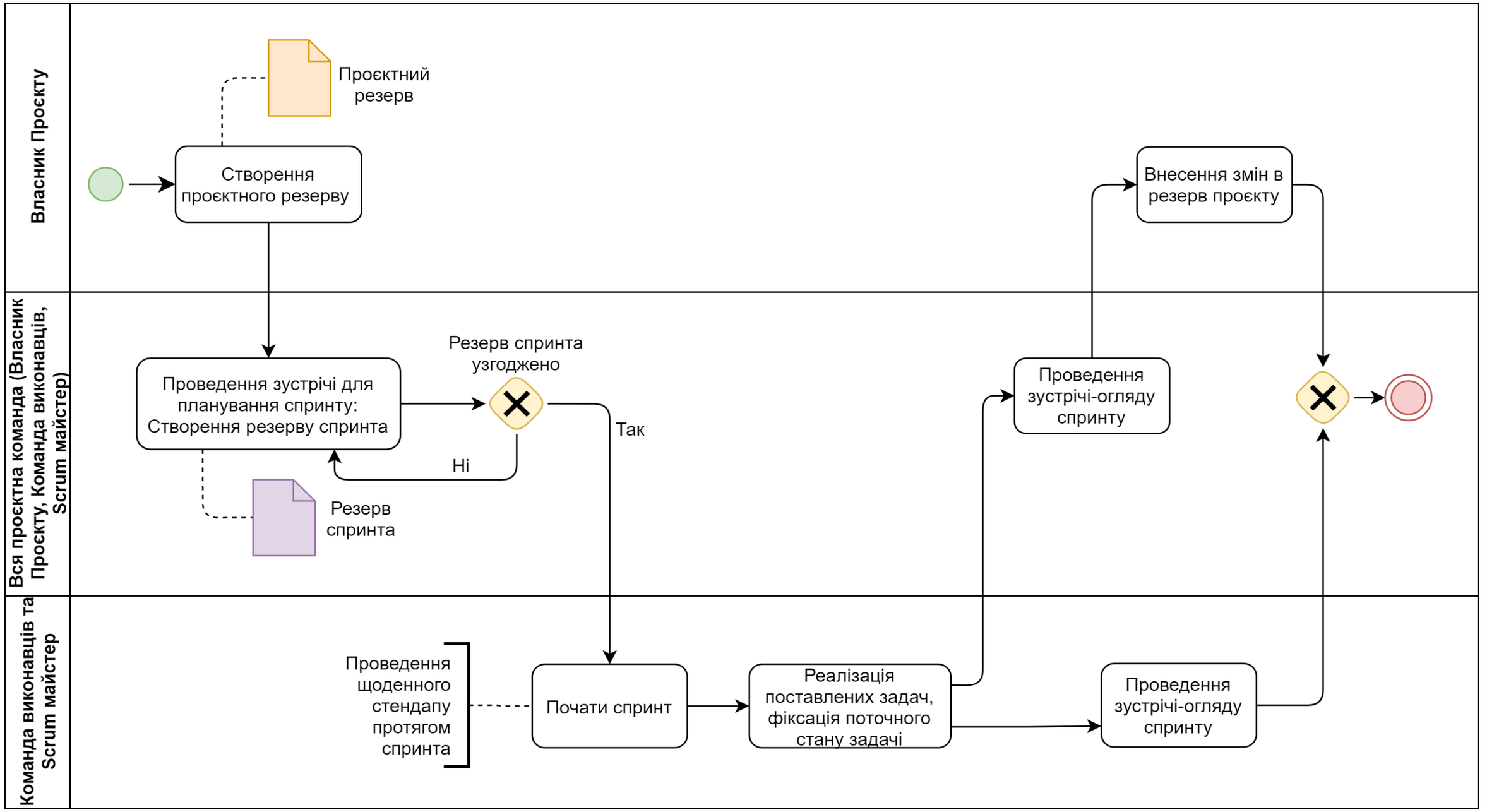
- функція ведення виконавців: користувач повинен мати можливість зареєструватись в системі, авторизуватись в системі;
- функція ведення проєктів: користувач повинен мати можливість додавати та видаляти проєкти;
- функція ведення задач: користувач повинен мати можливість додавати та видаляти задачі;

- функція ведення проєктних команд: користувач повинен мати можливість додавати та видаляти виконавців з проєктної команди;
- функція ведення задач виконавців: користувач повинен мати можливість додавати, редагувати та видаляти командні роботи;
- функція складання графіку виконання задач для кожного виконавця: система повинна надавати можливість користувачу генерувати план виконання робіт учасниками команди на основі кінцевих строків виконання робіт, кількості робіт, кількості учасників команди, завантаженості учасників команди;
- функція визначення загальної кількості виконаних задач за період часу, кількості задач, закінчених невчасно;
- функція фіксації виконавцями поточного стану задачі та скільки часу було витрачено на задачу: користувач повинен мати можливість позначити задачу як розпочату, відхилену, завершену, та вказати кількість часу, який було витрачено на задачу;
- функція обробки введення некоректних даних.

Графічний матеріал
до дипломного проєкту

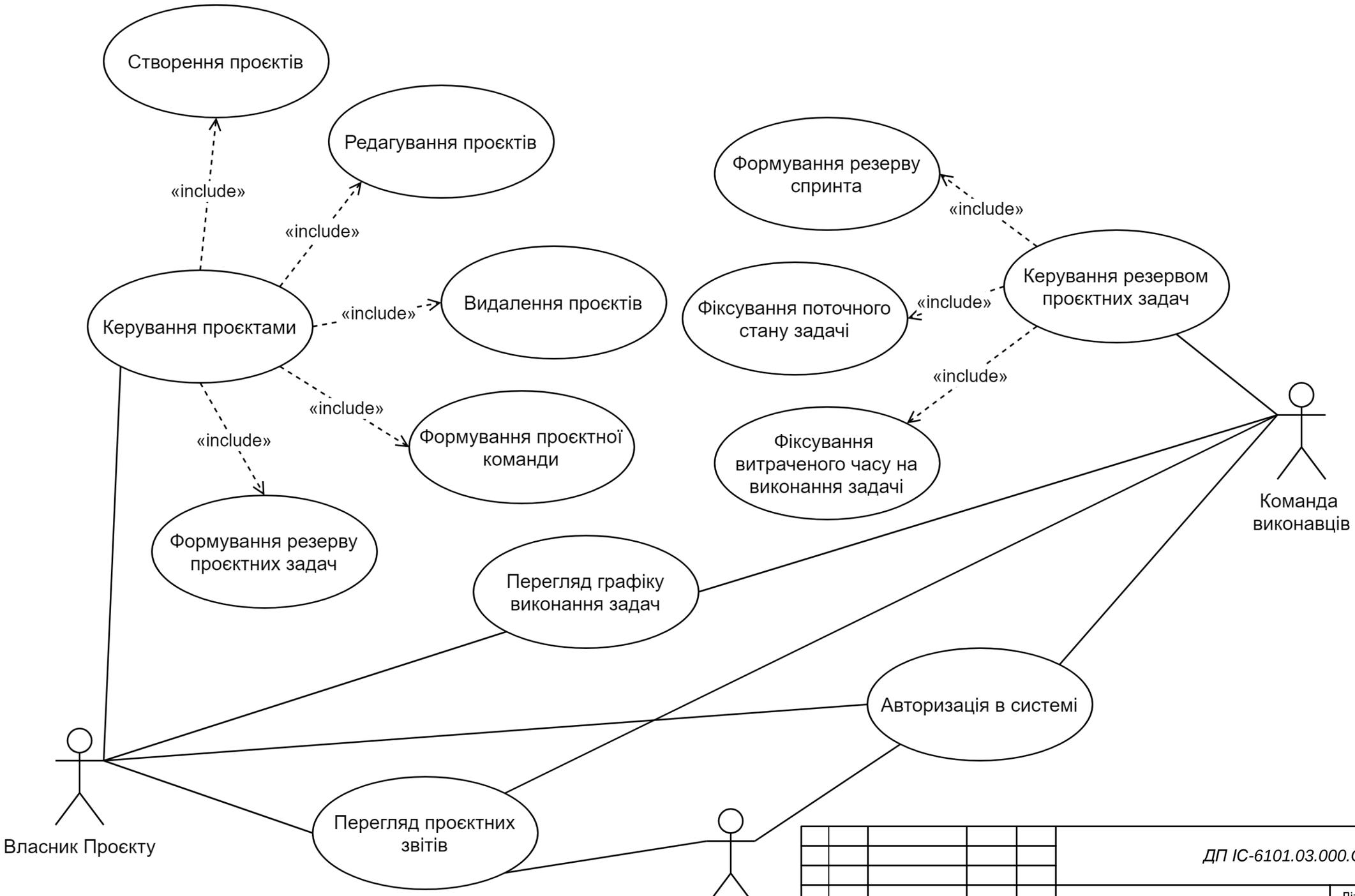
на тему: Комплекс задач з підтримки процесу управління командою виконавців

Київ – 2020 року



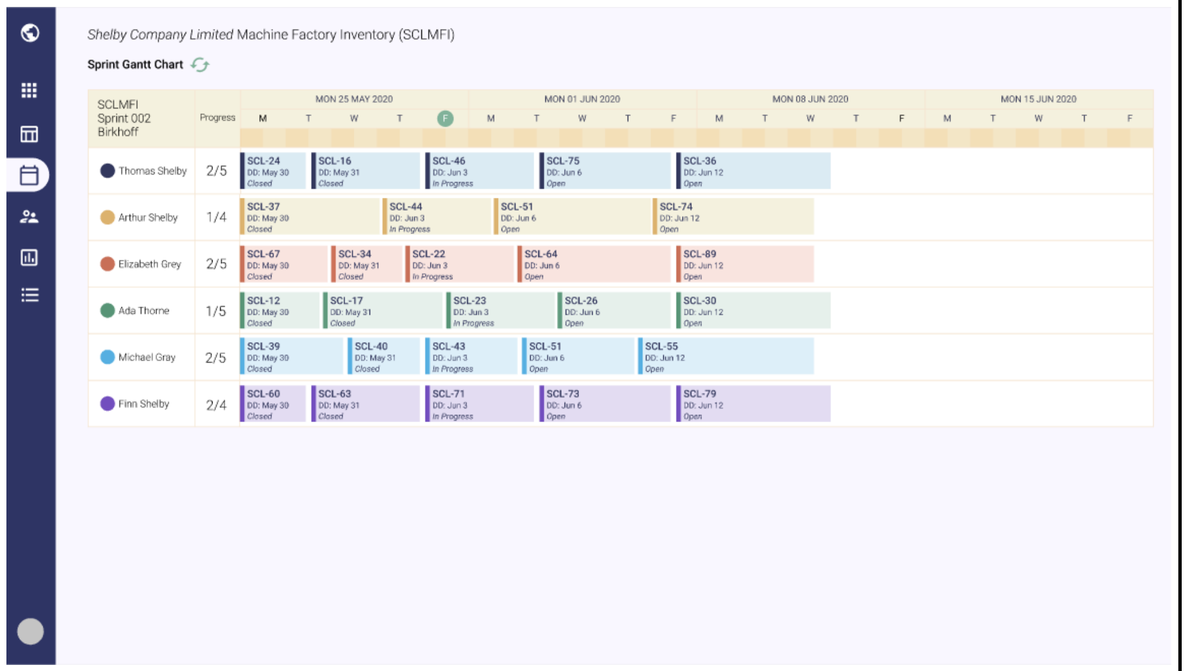
					ДП ІС-6101.02.000.ССД	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	
Розробив		Авраменко К.А.				Аркуш 1 / Аркушів 1
Перевірив		Жданова О.Г.				
Т. кон.						
Н. кон.		Проскура С.Л.			Комплекс задач з підтримки процесу управління командою виконавців	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63
Затвердив		Жданова О.Г.				

ДП ІС-6101.03.000.ССВ



					ДП ІС-6101.03.000.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна варіантів використання*	Літера / Маса / Масштаб
Розробив		Авраменко К.А.				
Перевірив		Жданова О.Г.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Проскура С.Л.			*Комплекс задач з підтримки процесу управління командою виконавців*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*
Затвердив		Жданова О.Г.				

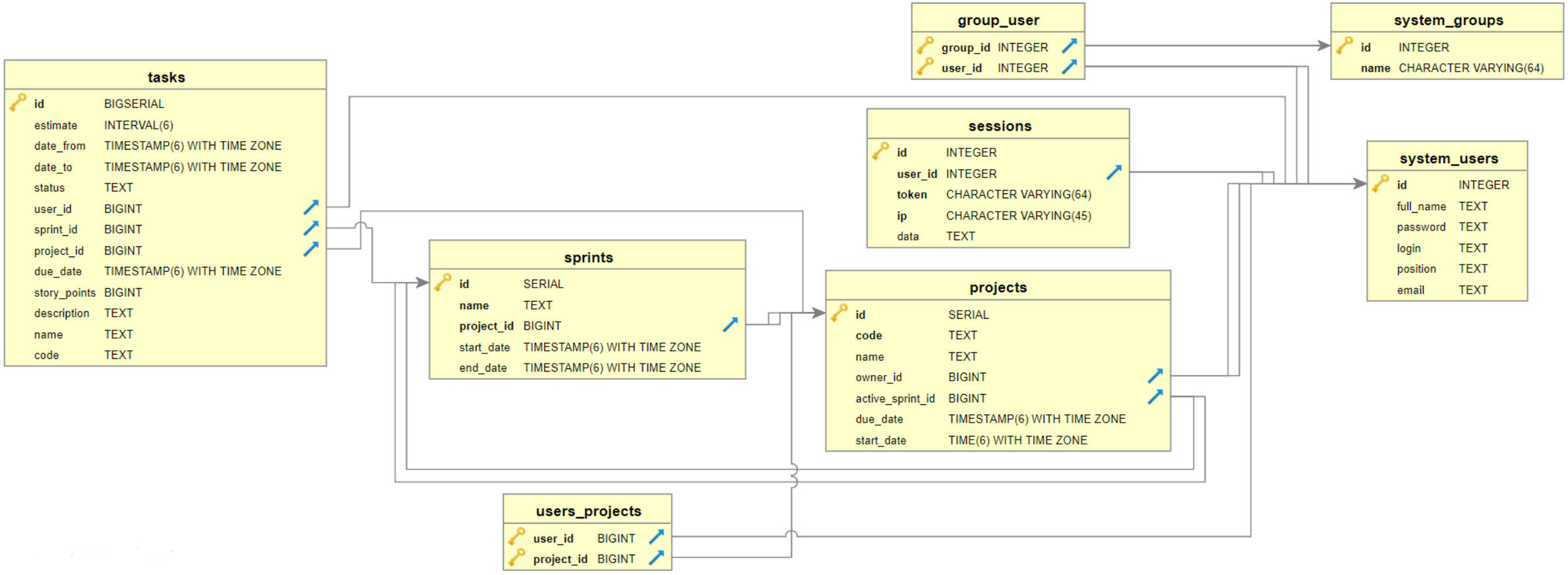
Звіт з планом виконання задач для команди виконавців



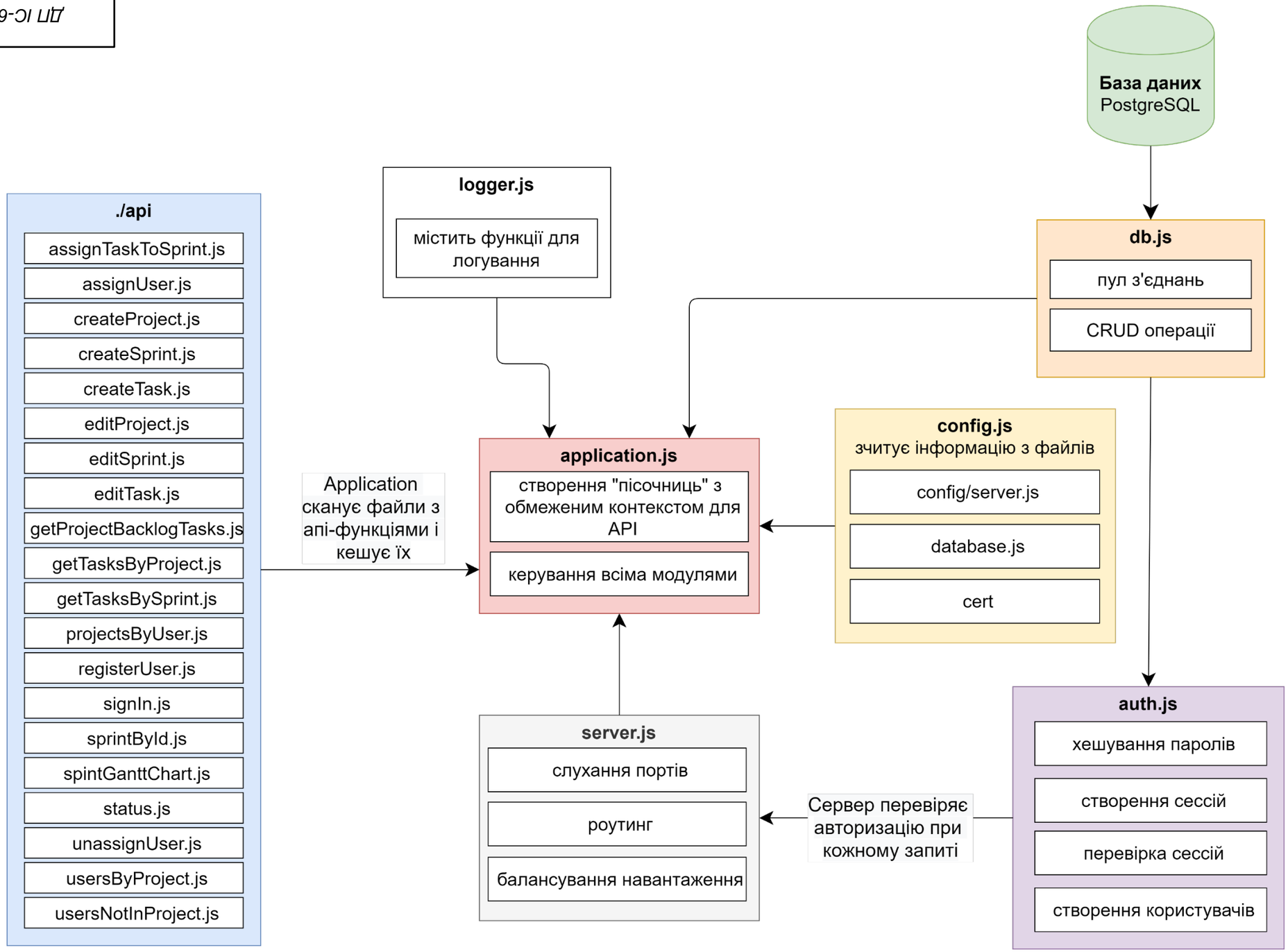
Звіт з результатами спринта

					ДП ІС-6101.04.000.КЕ	
Зм.	Арк.	№ докум.	Підп.	Дата	Креслення вигляду екранних форм	Лит. / Маса / Масштаб
Розроб.		Авраменко К.А.				
Перев.		Жданова О.Г.				Аркуш 1 / Аркушів 1
Т. Кон.						
Н. Кон.		Проскура С.Л.			Комплекс задач з підтримки процесу управління командою виконавців	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63
Затв.		Жданова О.Г.				

ДП ІС-6101.05.000.СБД

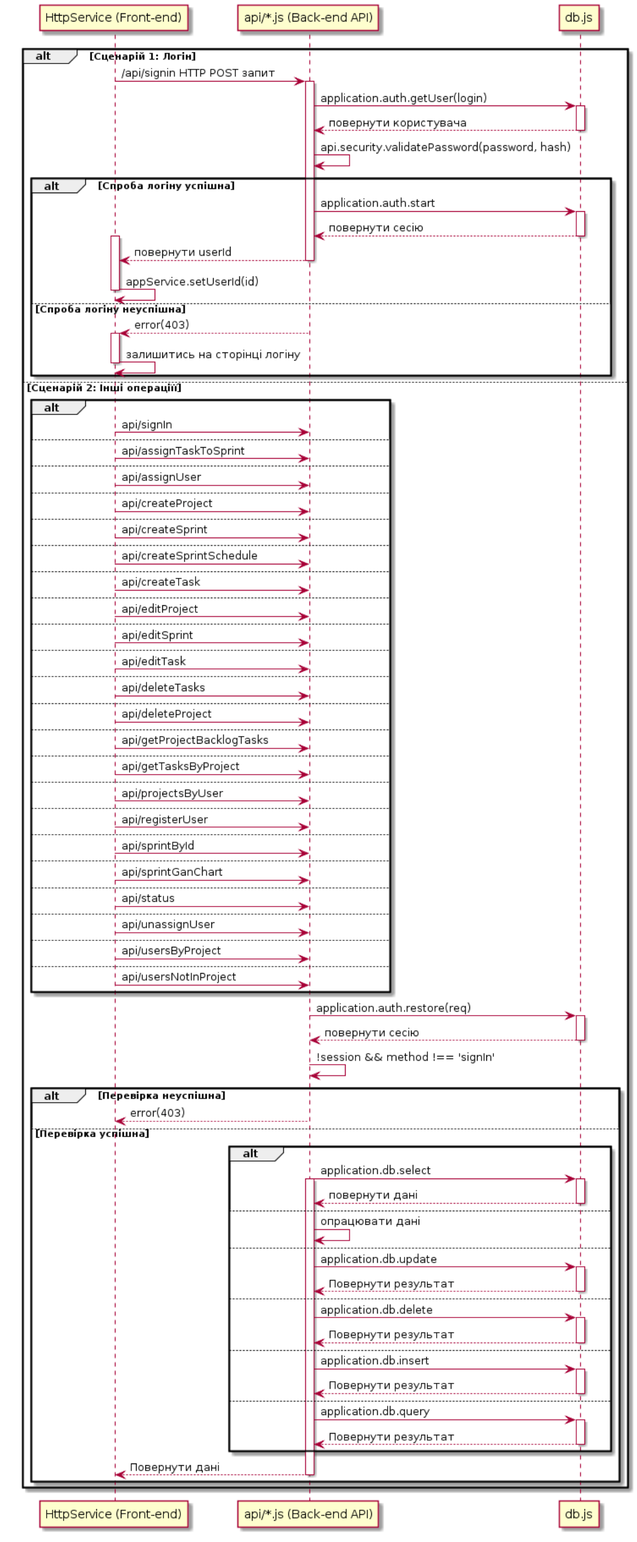


					ДП ІС-6101.05.000.СБД		
					Структура бази даних	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Авраменко К.А.				Аркуш 1	Аркушів 1
Перевірив		Жданова О.Г.					
Т. кон.							
Н. кон.		Проскура С.Л.			*Комплекс задач з підтримки процесу управління командою виконавців*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*	
Затвердив		Жданова О.Г.					



					ДП ІС-6101.06.000.САС	
					Схема структурна частини АС	Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Авраменко К.А.				
Перевірив		Жданова О.Г.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Проскура С.Л.			*Комплекс задач з підтримки процесу управління командою виконавців*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*
Затвердив		Жданова О.Г.				

ДП ІС-6101.07.000.ССП



Зм.	Арк.	№ докум.	Підп.	Дата	ДП ІС-6101.07.000.ССП		
Розроб.		Авраменко К.А.			Схема структурна послідовності	Лит.	Маса / Масштаб
Перев.		Жданова О.Г.					
Т. Кон.						Аркуш 1	Аркушів 1
Н. Кон.		Проскура С.Л.			Комплекс задач з підтримки процесу управління командою виконавців	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63	
Затв.		Жданова О.Г.					